Work Contracts

Rethinking Task Based Concurrency and Parallelism for Low Latency C++

MICHAEL A MANISCALCO

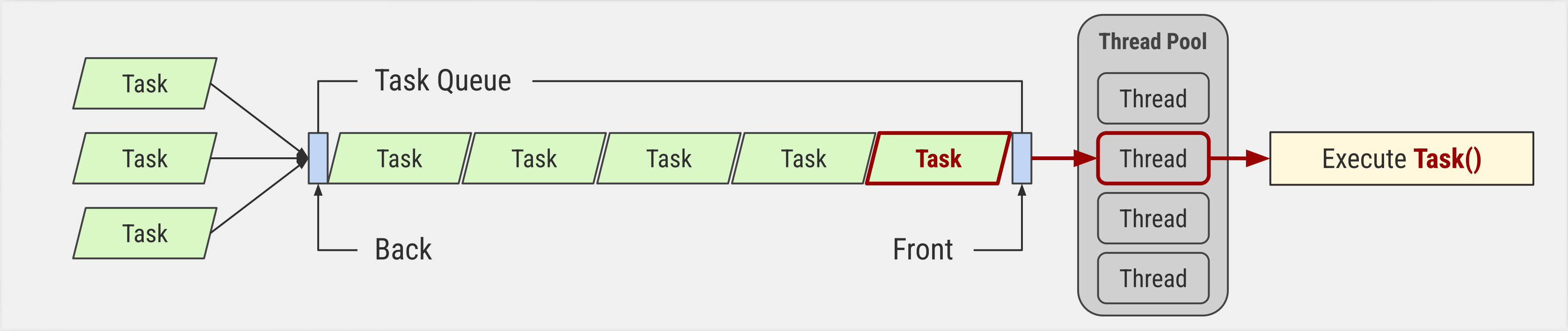
Cppcon
The C++ Conference

2024
September 15 - 20

> “We cannot solve our problems with the same thinking we used when we created them.”
>
> - Albert Einstein

So what is there to
Rethink?

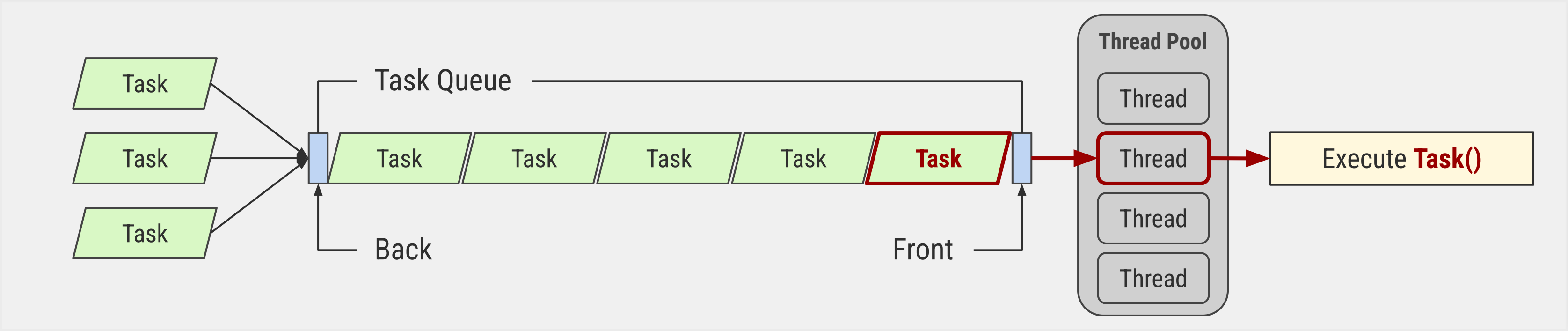
Rethinking: Task Queues



Problem #1 - Task Queues Do Not Scale Well:

- Contention:
  - Even the most meticulously designed lock-free queues experience a significant drop in performance as the number of threads increases.
- Multiple sub-queues can be used to mitigate contention but this introduces a myriad of new problems:
  - Task starvation
  - Load balancing
  - Forfeits strict FIFO behaviour
  - Increases memory footprint (or requires allocations)
  - Terrible task selection “fairness”

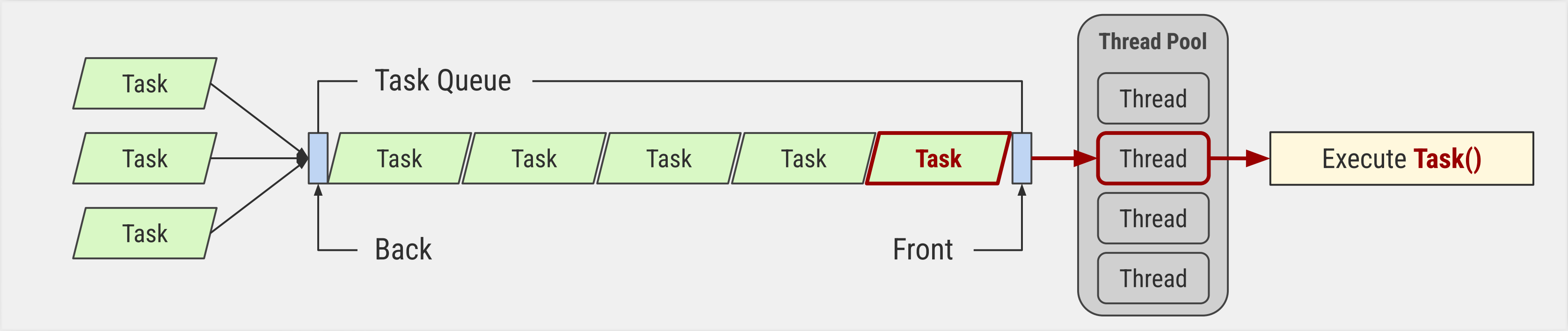
Rethinking: Task Queues



Problem #2 - No Inherent Support For Prioritization:

- Priority queues address this but also come with new problems:
    - Increased complexity
    - Further degrade performance
    - Task starvation
    - Task aging
- Multiple queues for different priority also works but:
    - Scheduling
    - Task starvation, load balancing, work stealing

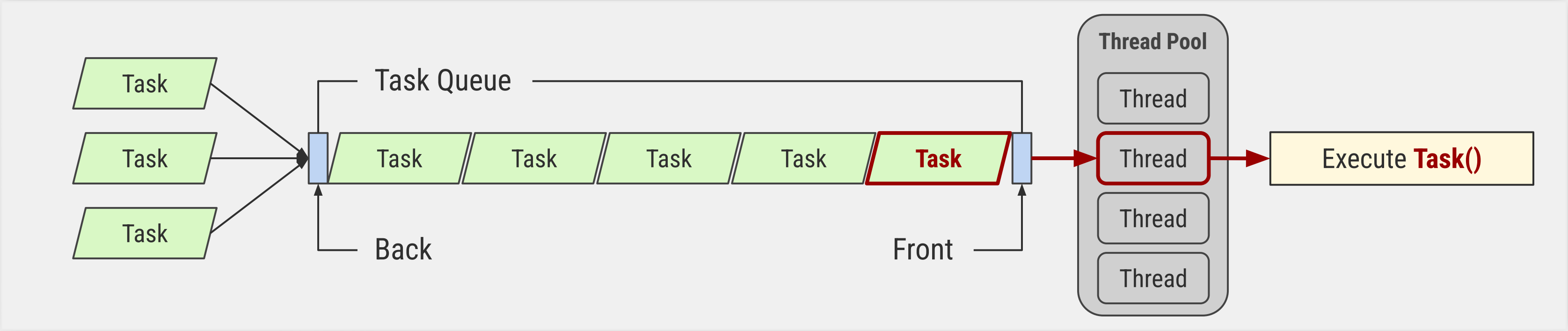
Rethinking: Task Queues



Problem #3 - Encourages Tight Coupling of Data and Logic:

- Queues are great for data:
  - FIFO for data is usually desirable
  - We usually want to preserve the order of our data
- Queues for logic (work) is not a good idea:
  - We couple logic and data together as tasks because:
    - Task Queues typically contain more than one type of task (different types of logic)
    - By the time a task reaches the front of the queue it is impossible to know which logic to apply to the data unless coupled together
  - We could decouple logic from data but:
    - Then we would require a unique queue for each type of logic
    - But multiple queues brings other headaches such as scheduling, prioritization, etc, as enumerated previously

Rethinking: Task Queues



Summary:

- Queues do not scale well:
  - True even for the best lock free implementations
- Does not support prioritization:
  - Available options to mitigate this shortcoming introduce a myriad of other shortcomings which further complicate issues
- By their nature, queues bring tasks to threads:
  - Requiring tasks to convey both data and logic
  - Often leading to task class hierarchies, pointers to base classes, pools and allocators

What is the

Alternative?

About Me:

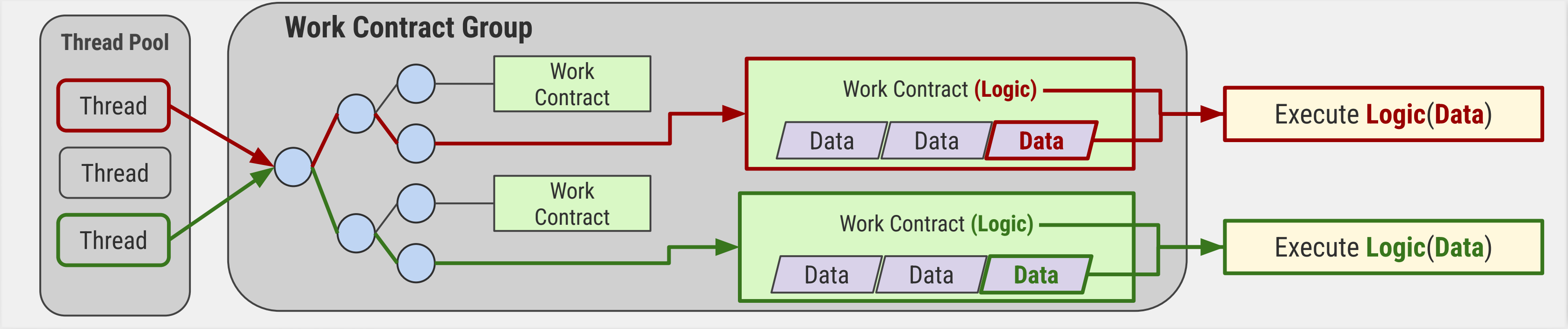
- Michael A Maniscalco
  - Software Architect and Principal Developer at Lime Trading
  - We develop low latency market data and trading software for use in HFT
- Personal
  - github.com/buildingcpp
    - Work Contracts, Networking, Messaging, Glimpse Instrumentation, etc …
  - github.com/michaelmaniscalco
    - Algorithms and Data Compression
    - M99, M03, RLE-EXP, MSufSort

LIME
TRADING

What is the

Alternative?

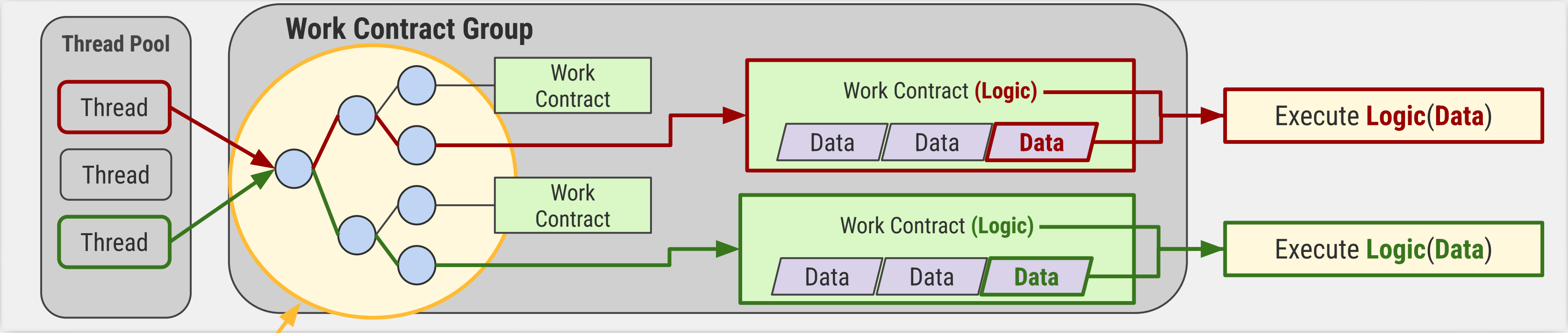
Alternative: Work Contracts



Work Contracts (Overview):

- A **Work Contract Group** contains:
  - An array of **Work Contracts** (each with their own logic and, if needed, data, queue<data>, etc)
  - A **Signal Tree** (which has as many leaf nodes as there are work contracts in the group)
- Threads are brought to the “task” rather than the “task” being brought to the threads (as is done with Task Queues)
  - Minimize thread contention because threads do not compete for the same Work Contract (task).

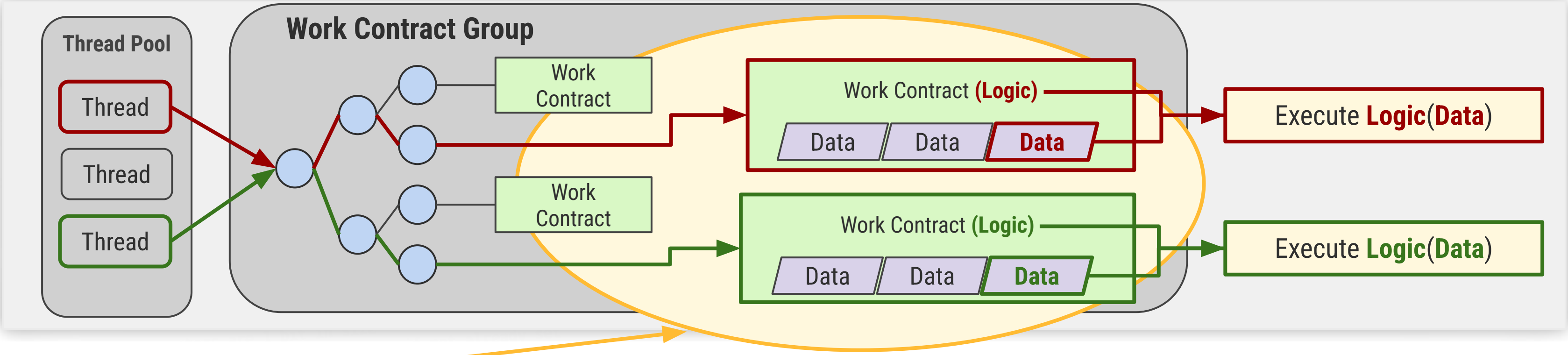
Alternative: Work Contracts



The Signal Tree:

- Lock Free with most operations also wait free
  - Allows MT traversal without locks and therefore parallel task execution with near zero contention between threads
- Excellent Scalability:
  - Over 40x higher throughput than the fastest MPMC queue at scale
  - Over 100x higher throughput than the average MPMC queue at scale
  - Approximately 1/2N memory requirement (N = number of nodes)

Alternative: Work Contracts



Work Contract:

- Enhanced “Tasks” separating data from logic:
  - Contain its own logic
  - Asynchronous execution
  - Recurrening execution
  - Accesses its own data (user defined ingress)
  - Guaranteed single threaded execution
  - Supports optional asynchronous destruction logic

Resources:

Source Code:

github.com/buildingcpp/work_contract

This Talk:

cppcon2024.sched.com

Contact:

wc@michael-maniscalco.com

Lime Trading:

Lime.co

LIME
TRADING

Benchmarks:

Test Environment

- Ubuntu 24.04
- GCC 13.2.0
- 13th Gen Intel(R) Core(TM) i9-13900HX
  - Hyperthreading disabled
  - All 16 “efficient” cores isolated

Three established MPMC queues were benchmarked along with Work Contracts:

- Boost “lock free”
- TBB “concurrent_queue”
- MoodyCamel “ConcurrentQueue”
- Work Contracts

Caution:
Slideware Ahead

"Ever since I started using Slideware, I can give hour long presentations and no one even mentions all of my compiler errors. Thanks, Slideware!"

Warning: In rare cases, exposure to slideware can lead to side effects such as drowsiness, glazed eyes, a sudden desire to migrate to a neighboring talk, nausea, programmer burnout, or the belief that the presenter doesn't understand the first thing about move semantics. If you experience any of these symptoms, a persistent sense of boredom, or the sudden urge to analyze patterns in the conference room carpet, then ask your project manager if slideware is right for you.

The Benchmark:

```
template <typename queue_type, typename task_type>
void bench(task_type && task, auto numThreads, auto numTasks)
{
    queue_type<task_type> tasks(numTasks);
    for (auto & t : tasks)
        t = task;
    std::jthread threads(numThreads);
    for (auto & thread : threads)
        thread = std::move([&](){while (!start); while (!stop){auto t = tasks.pop(); t(); tasks.push(t);}});
    start_timer();
    std::this_thread::sleep_for(test_duration);
    stop_timer();
    calculate_result();
}


template <typename task_type>
void bench(task_type && task)
{
    static auto constexpr numTasks = ((1 << 10) * 16);
    for (auto numThreads = 2; numThreads <= 16; ++numThreads)
    {
        bench<boost_lockfree>(std::forward<task_type>(task), numThreads, numTasks);
        bench<tbb_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<moodycamel_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<work_contract>(std::forward<task_type>(task), numThreads, numTasks);
    }
}
```

The Benchmark:

```
template <typename queue_type, typename task_type>
void bench(task_type && task, auto numThreads, auto numTasks)
{
    queue_type<task_type> tasks(numTasks);
    for (auto & t : tasks)
        t = task;
    std::jthread threads(numThreads);
    for (auto & thread : threads)
        thread = std::move([&](){while (!start); while (!stop){auto t = tasks.pop(); t(); tasks.push(t);}});
    start_timer();
    std::this_thread::sleep_for(test_duration);
    stop_timer();
    calculate_result();
}


template <typename task_type>
void bench(task_type && task)
{
    static auto constexpr numTasks = ((1 << 10) * 16);
    for (auto numThreads = 2; numThreads <= 16; ++numThreads)
    {
        bench<boost_lockfree>(std::forward<task_type>(task), numThreads, numTasks);
        bench<tbb_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<moodycamel_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<work_contract>(std::forward<task_type>(task), numThreads, numTasks);
    }
}
```

The Benchmark:

```
template <typename queue_type, typename task_type>
void bench(task_type && task, auto numThreads, auto numTasks)
{
    queue_type<task_type> tasks(numTasks);
    for (auto & t : tasks)
        t = task;
    std::jthread threads(numThreads);
    for (auto & thread : threads)
        thread = std::move([&](){while (!start); while (!stop){auto t = tasks.pop(); t(); tasks.push(t);}});
    start_timer();
    std::this_thread::sleep_for(test_duration);
    stop_timer();
    calculate_result();
}


template <typename task_type>
void bench(task_type && task)
{
    static auto constexpr numTasks = ((1 << 10) * 16);
    for (auto numThreads = 2; numThreads <= 16; ++numThreads)
    {
        bench<boost_lockfree>(std::forward<task_type>(task), numThreads, numTasks);
        bench<tbb_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<moodycamel_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<work_contract>(std::forward<task_type>(task), numThreads, numTasks);
    }
}
```

The Benchmark:

```
template <typename queue_type, typename task_type>
void bench(task_type && task, auto numThreads, auto numTasks)
{
    queue_type<task_type> tasks(numTasks);
    for (auto & t : tasks)
        t = task;
    std::jthread threads(numThreads);
    for (auto & thread : threads)
        thread = std::move([&](){while (!start); while (!stop){auto t = tasks.pop(); t(); tasks.push(t);}});
    start_timer();
    std::this_thread::sleep_for(test_duration);
    stop_timer();
    calculate_result();
}


template <typename task_type>
void bench(task_type && task)
{
    static auto constexpr numTasks = ((1 << 10) * 16);
    for (auto numThreads = 2; numThreads <= 16; ++numThreads)
    {
        bench<boost_lockfree>(std::forward<task_type>(task), numThreads, numTasks);
        bench<tbb_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<moodycamel_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<work_contract>(std::forward<task_type>(task), numThreads, numTasks);
    }
}
```

The Benchmark:

```
template <typename queue_type, typename task_type>
void bench(task_type && task, auto numThreads, auto numTasks)
{
    queue_type<task_type> tasks(numTasks);
    for (auto & t : tasks)
        t = task;
    std::jthread threads(numThreads);
    for (auto & thread : threads)
        thread = std::move([&](){while (!start); while (!stop){auto t = tasks.pop(); t(); tasks.push(t);}});
    start_timer();
    std::this_thread::sleep_for(test_duration);
    stop_timer();
    calculate_result();
}


template <typename task_type>
void bench(task_type && task)
{
    static auto constexpr numTasks = ((1 << 10) * 16);
    for (auto numThreads = 2; numThreads <= 16; ++numThreads)
    {
        bench<boost_lockfree>(std::forward<task_type>(task), numThreads, numTasks);
        bench<tbb_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<moodycamel_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<work_contract>(std::forward<task_type>(task), numThreads, numTasks);
    }
}
```

The Benchmark:

```
template <typename queue_type, typename task_type>
void bench(task_type && task, auto numThreads, auto numTasks)
{
    queue_type<task_type> tasks(numTasks);
    for (auto & t : tasks)
        t = task;
    std::jthread threads(numThreads);
    for (auto & thread : threads)
        thread = std::move([&](){while (!start); while (!stop){auto t = tasks.pop(); t(); tasks.push(t);}});
    start_timer();
    std::this_thread::sleep_for(test_duration);
    stop_timer();
    calculate_result();
}


template <typename task_type>
void bench(task_type && task)
{
    static auto constexpr numTasks = ((1 << 10) * 16);
    for (auto numThreads = 2; numThreads <= 16; ++numThreads)
    {
        bench<boost_lockfree>(std::forward<task_type>(task), numThreads, numTasks);
        bench<tbb_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<moodycamel_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<work_contract>(std::forward<task_type>(task), numThreads, numTasks);
    }
}
```

The Benchmark:

```
template <typename queue_type, typename task_type>
void bench(task_type && task, auto numThreads, auto numTasks)
{
    queue_type<task_type> tasks(numTasks);
    for (auto & t : tasks)
        t = task;
    std::jthread threads(numThreads);
    for (auto & thread : threads)
        thread = std::move([&](){while (!start); while (!stop){auto t = tasks.pop(); t(); tasks.push(t);}});
    start_timer();
    std::this_thread::sleep_for(test_duration);
    stop_timer();
    calculate_result();
}


template <typename task_type>
void bench(task_type && task)
{
    static auto constexpr numTasks = ((1 << 10) * 16);
    for (auto numThreads = 2; numThreads <= 16; ++numThreads)
    {
        bench<boost_lockfree>(std::forward<task_type>(task), numThreads, numTasks);
        bench<tbb_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<moodycamel_queue>(std::forward<task_type>(task), numThreads, numTasks);
        bench<work_contract>(std::forward<task_type>(task), numThreads, numTasks);
    }
}
```

The Benchmark:

```
template <typename queue_type, typename task_type>
void bench(task_type && task,
{
    queue_type<task_type> task
    for (auto & t : tasks)
        t = task;
    std::jthread threads(numTh
    for (auto & thread : threa
        thread = std::move([&]
    start_timer();
    std::this_thread::sleep_fo
    stop_timer();
    calculate_result();
}


template <typename task_type>
void bench(task_type && task)
{
    static auto constexpr numT
    for (auto numThreads = 2;
    {
        bench<boost_lockfree>(
        bench<tbb_queue>(std::
        bench<moodycamel_queue
        bench<work_contract>(s
    }
}
```

The “Task”:

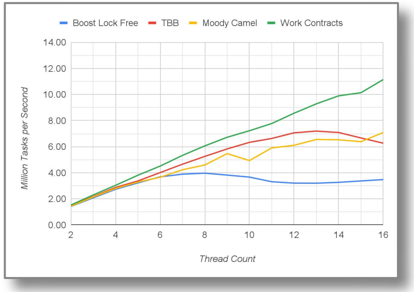
```
template <std::size_t N>
auto hash_task()
{
    static auto constexpr str = "guess what? chicken butt!";
    auto volatile n = 0;
    for (auto i = 0; i < N; ++i)
        n *= std::hash<std::string>()(str);
    return n;
};

using max_contention_task = hash_task<0>;       // duration = approx 1.5ns
using high_contention_task = hash_task<1>;      // duration = approx 17ns
using medium_contention_task = hash_task<64>;  // duration = approx 1.1µs
using low_contention_task = hash_task<256>;    // duration = approx 4.2µs

void bench()
{
    bench<low_contention_task>();
    bench<medium_contention_task>();
    bench<high_contention_task>();
    bench<max_contention_task>();
}
```

The “Task”:

```cpp
template <std::size_t N>
auto hash_task()
{
    static auto constexpr str = "guess what? chicken butt!";
    auto volatile n = 0;
    for (auto i = 0; i < N; ++i)
        n *= std::hash<std::string>()(str);
    return n;
};

using max_contention_task = hash_task<0>;      // duration = approx 1.5ns
using high_contention_task = hash_task<1>;     // duration = approx 17ns
using medium_contention_task = hash_task<64>;  // duration = approx 1.1µs
using low_contention_task = hash_task<256>;    // duration = approx 4.2µs

void bench()
{
    bench<low_contention_task>();
    bench<medium_contention_task>();
    bench<high_contention_task>();
    bench<max_contention_task>();
}
```

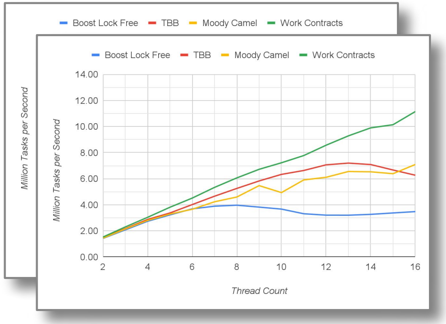
The “Task”:

```
template <std::size_t N>
auto hash_task()
{
    static auto constexpr str = "guess what? chicken butt!";
    auto volatile n = 0;
    for (auto i = 0; i < N; ++i)
        n *= std::hash<std::string>()(str);
    return n;
};

using max_contention_task = hash_task<0>;      // duration = approx 1.5ns
using high_contention_task = hash_task<1>;     // duration = approx 17ns
using medium_contention_task = hash_task<64>;  // duration = approx 1.1µs
using low_contention_task = hash_task<256>;    // duration = approx 4.2µs

void bench()
{
    bench<low_contention_task>();
    bench<medium_contention_task>();
    bench<high_contention_task>();
    bench<max_contention_task>();
}
```



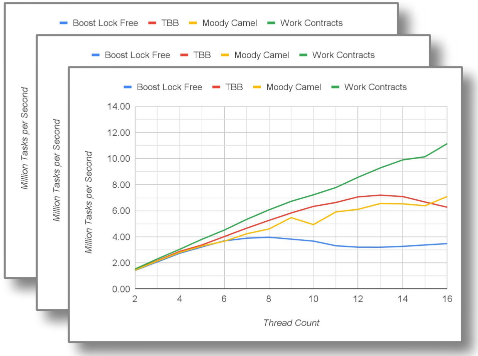
The “Task”:

```cpp
template <std::size_t N>
auto hash_task()
{
    static auto constexpr str = "guess what? chicken butt!";
    auto volatile n = 0;
    for (auto i = 0; i < N; ++i)
        n *= std::hash<std::string>()(str);
    return n;
};

using max_contention_task = hash_task<0>;       // duration = approx 1.5ns
using high_contention_task = hash_task<1>;      // duration = approx 17ns
using medium_contention_task = hash_task<64>;   // duration = approx 1.1µs
using low_contention_task = hash_task<256>;     // duration = approx 4.2µs

void bench()
{
    bench<low_contention_task>();
    bench<medium_contention_task>();
    bench<high_contention_task>();
    bench<max_contention_task>();
}
```



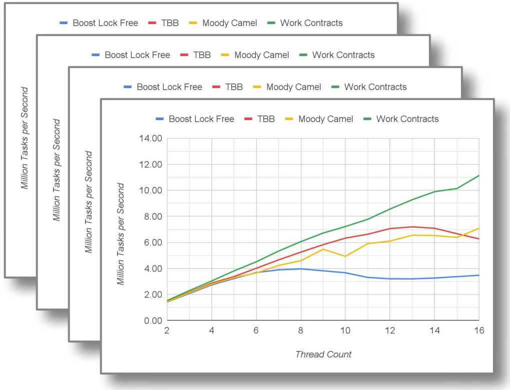
The “Task”:

```
template <std::size_t N>
auto hash_task()
{
    static auto constexpr str = "guess what? chicken butt!";
    auto volatile n = 0;
    for (auto i = 0; i < N; ++i)
        n *= std::hash<std::string>()(str);
    return n;
};

using max_contention_task = hash_task<0>;       // duration = approx 1.5ns
using high_contention_task = hash_task<1>;      // duration = approx 17ns
using medium_contention_task = hash_task<64>;  // duration = approx 1.1µs
using low_contention_task = hash_task<256>;    // duration = approx 4.2µs

void bench()
{
    bench<low_contention_task>();
    bench<medium_contention_task>();
    bench<high_contention_task>();
    bench<max_contention_task>();
}
```



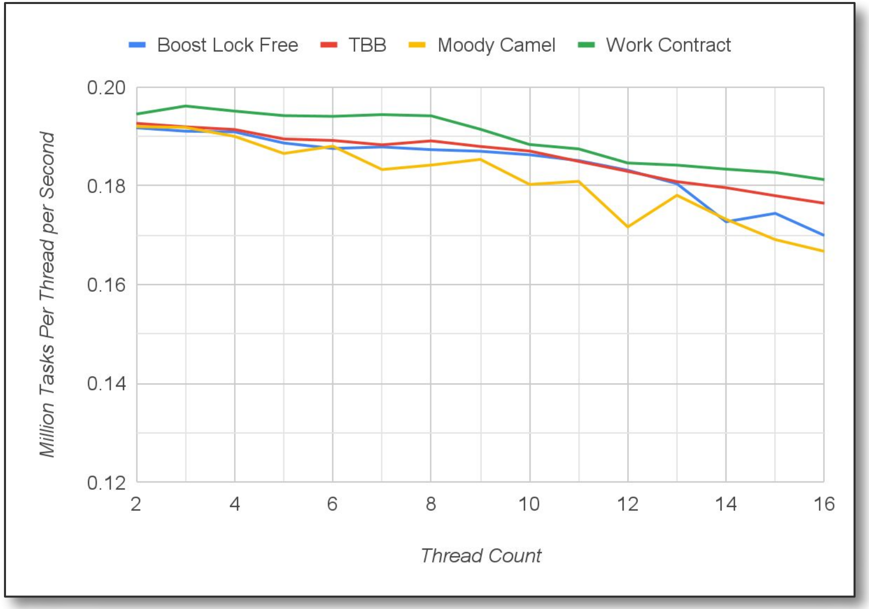
The “Task”:

```
template <std::size_t N>
auto hash_task()
{
    static auto constexpr str = "guess what? chicken butt!";
    auto volatile n = 0;
    for (auto i = 0; i < N; ++i)
        n *= std::hash<std::string>()(str);
    return n;
};

using max_contention_task = hash_task<0>;       // duration = approx 1.5ns
using high_contention_task = hash_task<1>;      // duration = approx 17ns
using medium_contention_task = hash_task<64>;  // duration = approx 1.1µs
using low_contention_task = hash_task<256>;    // duration = approx 4.2µs

void bench()
{
    bench<low_contention_task>();
    bench<medium_contention_task>();
    bench<high_contention_task>();
    bench<max_contention_task>();
}
```



Low Contention Benchmark

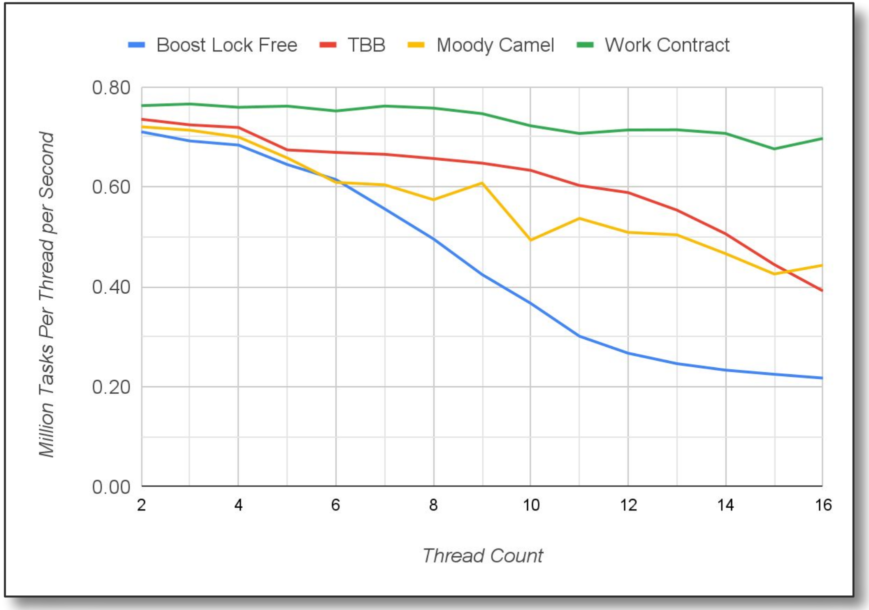
Approximate Task Duration: 4.200 µs



Coefficient of Variation at 16 Cores:	Boost Lock Free	TBB	Moody Camel	Work Contract
Task Fairness	< 0.01	< 0.01	4.75	0.06
Thread Fairness	0.01	< 0.01	0.09	< 0.01

Medium Contention Benchmark

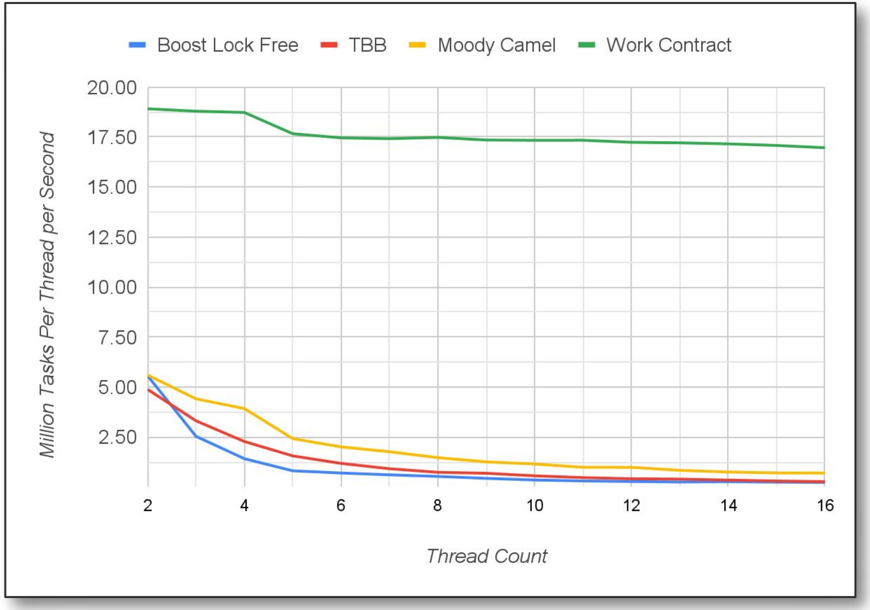
Approximate Task Duration: 1.1 μs



Coefficient of Variation at 16 Cores:	Boost Lock Free	TBB	Moody Camel	Work Contract
Task Fairness	< 0.01	< 0.01	1.02	0.03
Thread Fairness	0.04	0.04	0.21	0.02

High Contention Benchmark

Approximate Task Duration: 17 ns



Coefficient of Variation at 16 Cores:	Boost Lock Free	TBB	Moody Camel	Work Contract
Task Fairness	< 0.01	< 0.01	1.05	< 0.01
Thread Fairness	0.07	0.13	0.03	< 0.01

Maximum Contention Benchmark

Approximate Task Duration: 1.5 ns

Coefficient of Variation at 16 Cores:	Boost Lock Free	TBB	Moody Camel	Work Contract
Task Fairness	< 0.01	< 0.01	1.11	< 0.01
Thread Fairness	0.25	0.15	0.04	0.01

Where is this improvement coming from?

Signal Trees

The Signal Tree:

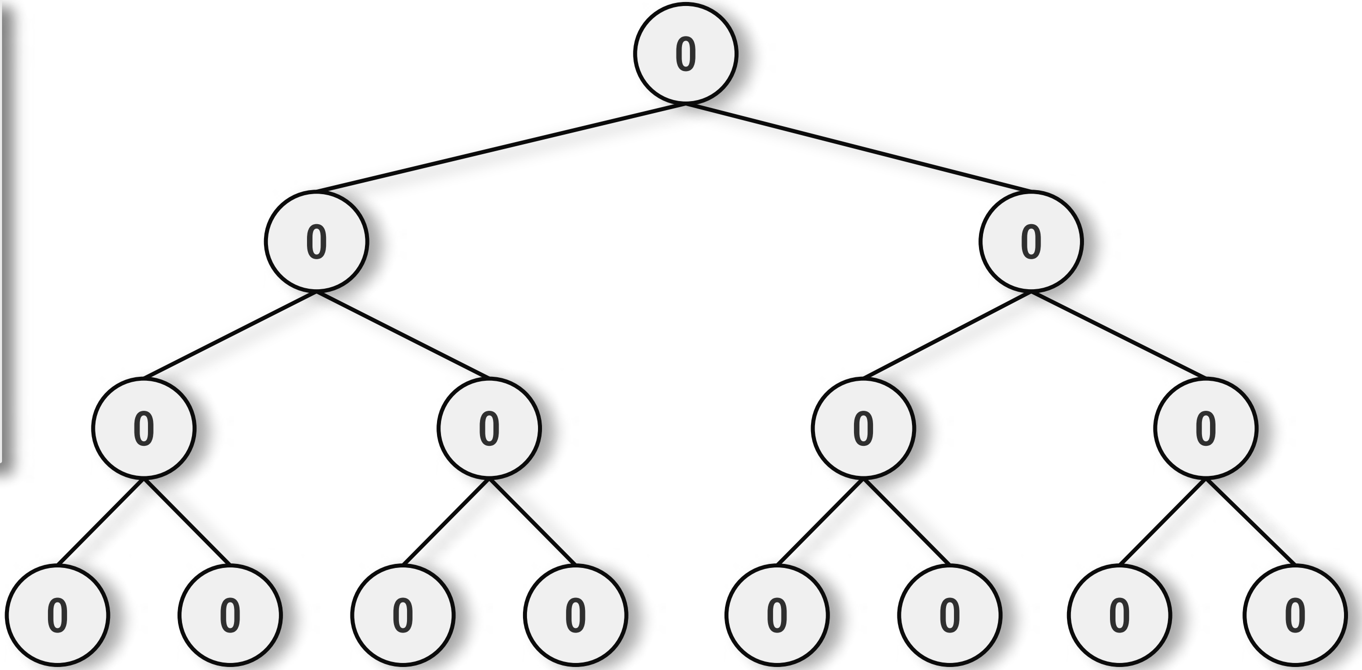
- Lock Free with most operations also wait free:
  - Allows MT traversal without locks and therefore parallel task execution with near zero contention between threads
- Excellent Scalability:
  - Over 40x higher throughput than the fastest MPMC queue at scale
  - Over 100x higher throughput than the average MPMC queue at scale
  - Approximately 1/2N memory requirement (N = number of nodes)
- Task Selection Bias:
  - Simple, efficient, technique allows for extremely fair task selection
  - Same technique can be used to create bias for task prioritization
  - Bias is per thread, not per tree, allowing each thread to dynamically change its bias between prioritization and fairness
  - Prioritization is opportunistic. Failure to find a high priority contract results in selection of a lower priority contract with zero additional overhead.

Properties of a Signal Tree

Perfect Binary Tree:
- Internal nodes have two children
- Leaf nodes all at the same level

Leaf Nodes Represent Signals:
- Zero indicates reset (off)
- One indicates set (on)

Child Sum Property:
- Internal nodes are sum of children

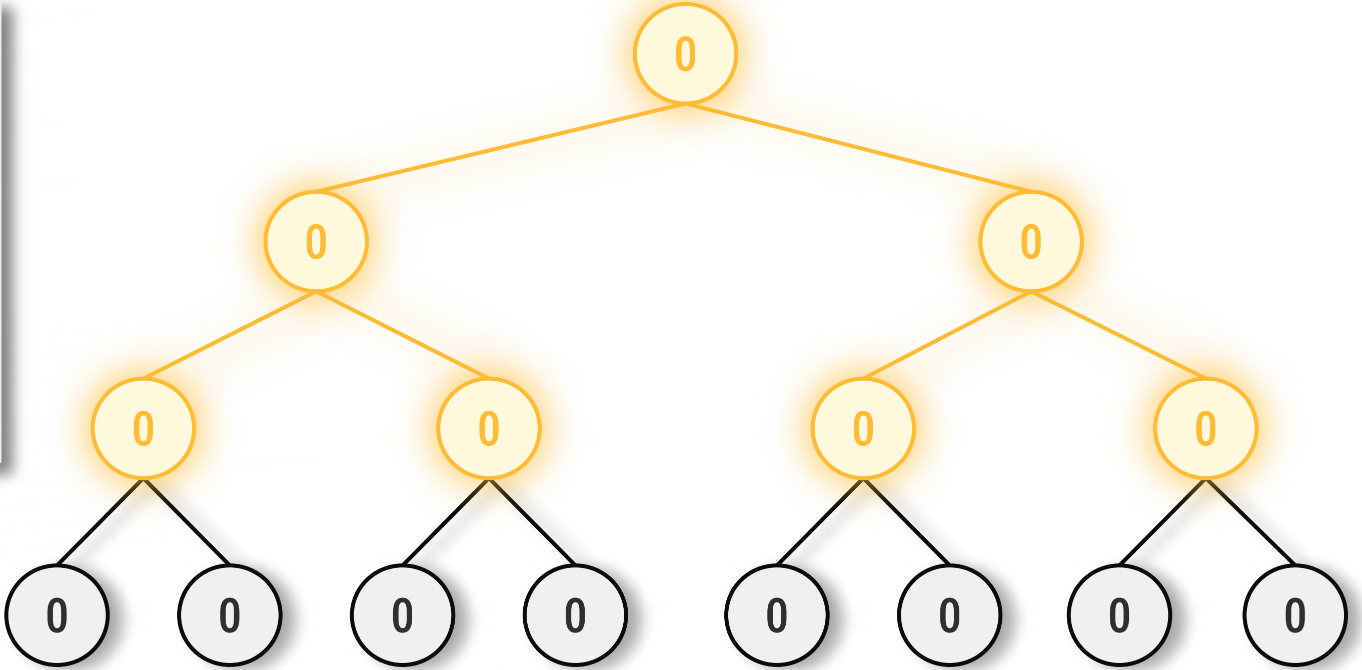


Properties of a Signal Tree

Perfect Binary Tree:
- Internal nodes have two children
- Leaf nodes all at the same level

Leaf Nodes Represent Signals:
- Zero indicates reset (off)
- One indicates set (on)

Child Sum Property:
- Internal nodes are sum of children



Properties of a Signal Tree

Perfect Binary Tree:

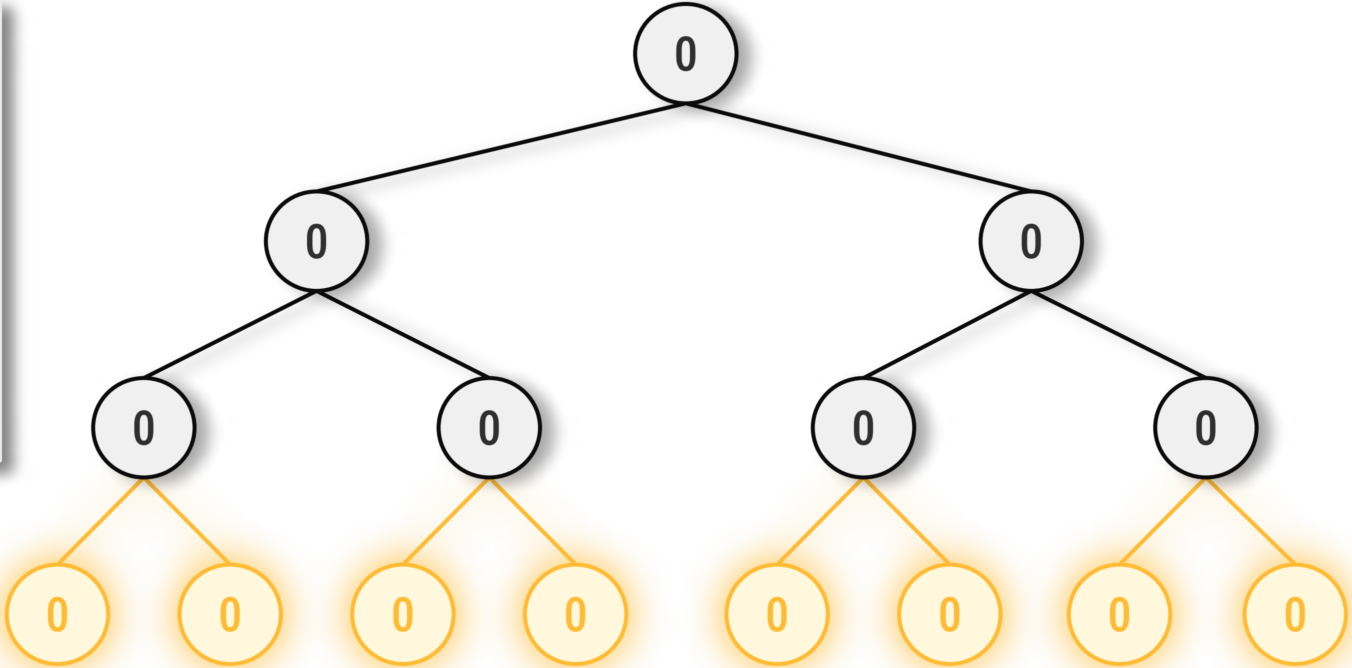
- Internal nodes have two children
- Leaf nodes all at the same level

Leaf Nodes Represent Signals:

- Zero indicates reset (off)
- One indicates set (on)

Child Sum Property:

- Internal nodes are sum of children



Properties of a Signal Tree

Perfect Binary Tree:

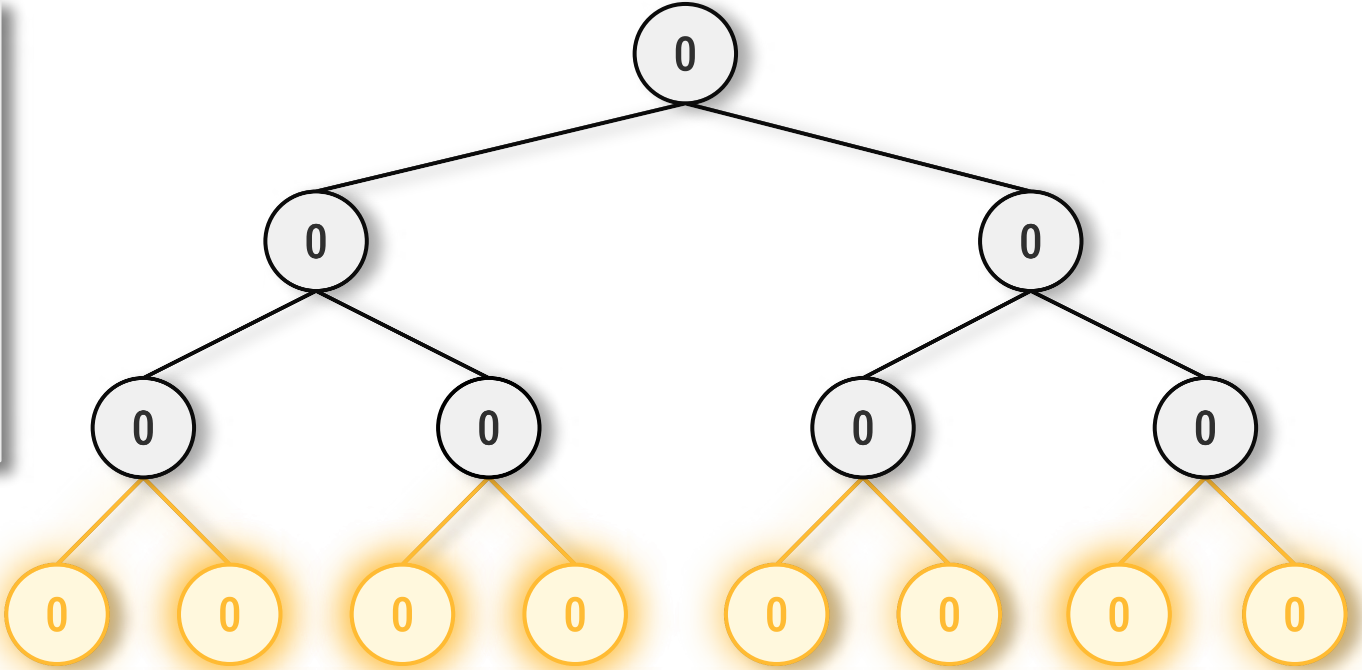
- Internal nodes have two children
- Leaf nodes all at the same level

Leaf Nodes Represent Signals:

- **Zero indicates reset**
- **One indicates set**

Child Sum Property:

- Internal nodes are sum of children



Properties of a Signal Tree

Perfect Binary Tree:

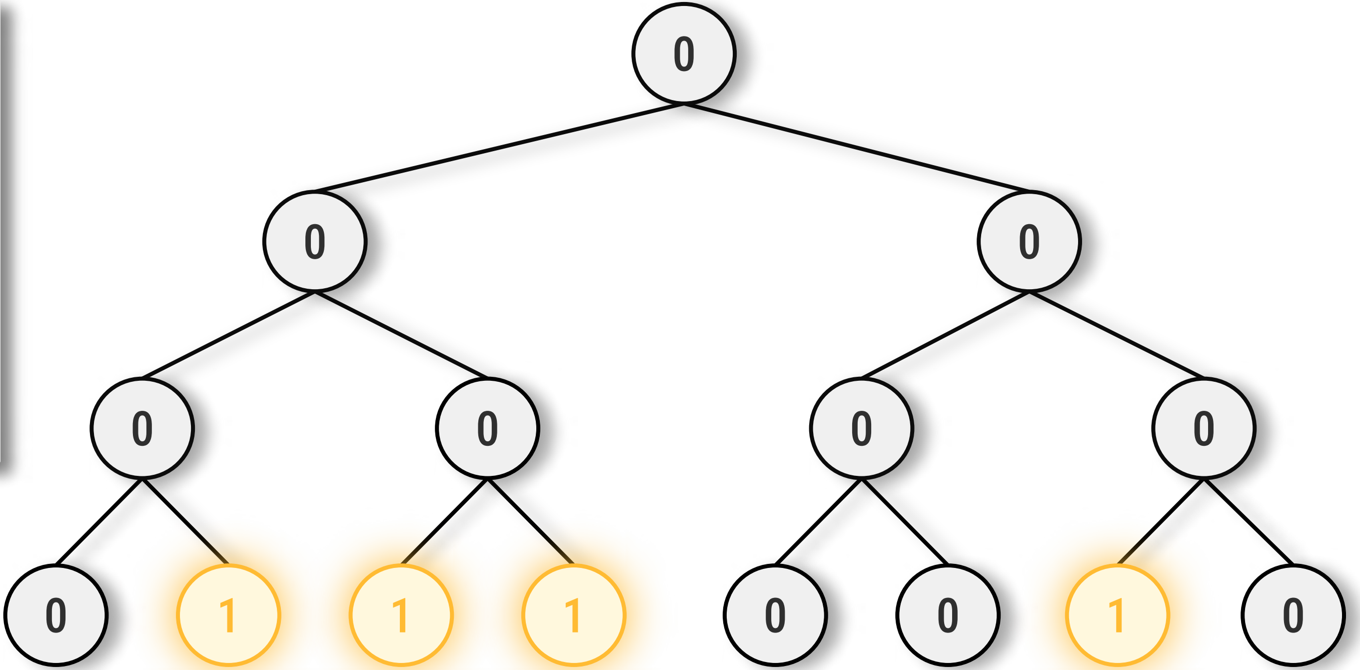
- Internal nodes have two children
- Leaf nodes all at the same level

Leaf Nodes Represent Signals:

- **Zero indicates reset**
- **One indicates set**

Child Sum Property:

- Internal nodes are sum of children

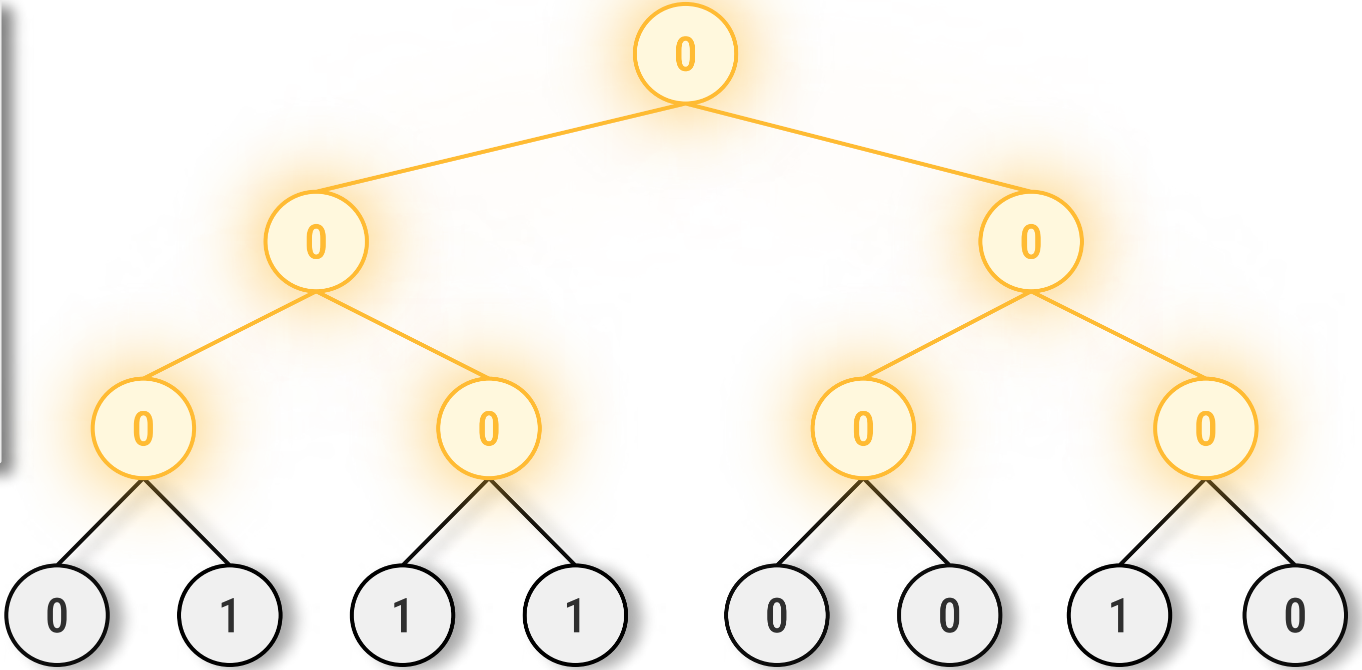


Properties of a Signal Tree

Perfect Binary Tree:
- Internal nodes have two children
- Leaf nodes all at the same level

Leaf Nodes Represent Signals:
- Zero indicates reset (off)
- One indicates set (on)

Child Sum Property:
- **Internal nodes are sum of children**



Properties of a Signal Tree

Perfect Binary Tree:

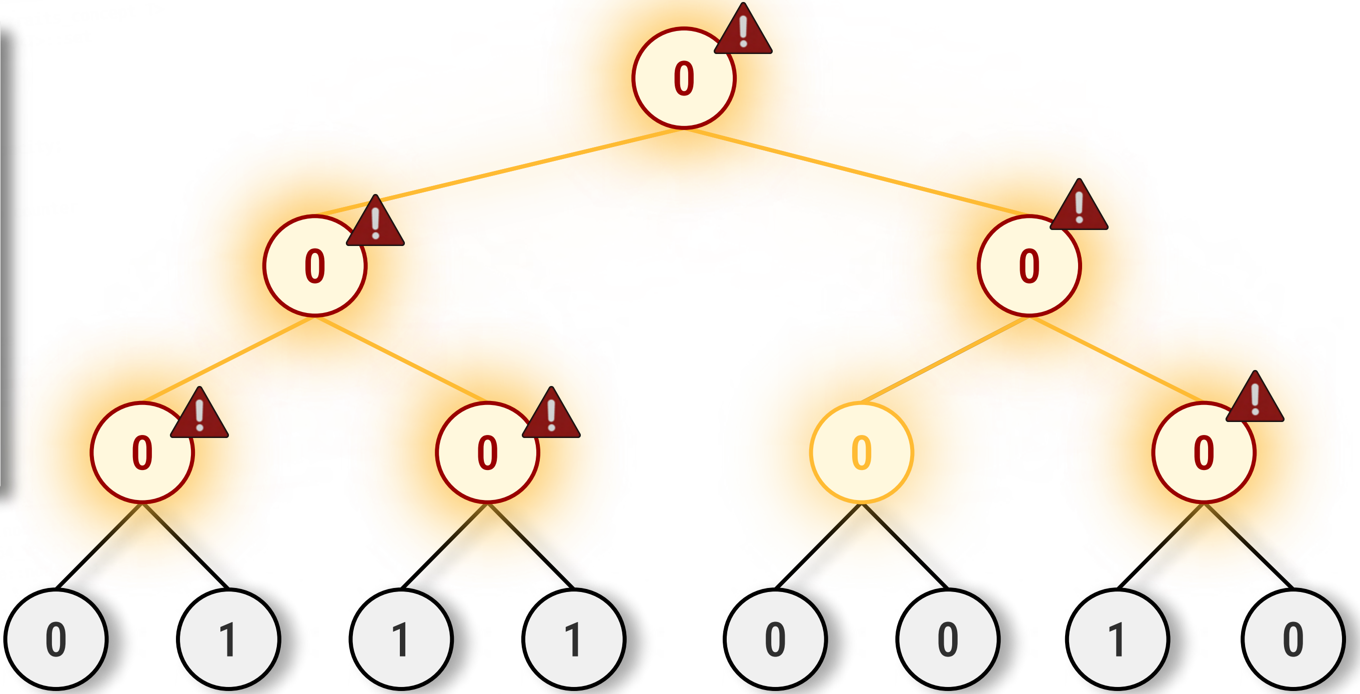
- Internal nodes have two children
- Leaf nodes all at the same level

Leaf Nodes Represent Signals:

- Zero indicates reset (off)
- One indicates set (on)

Child Sum Property:

- **Internal nodes are sum of children**

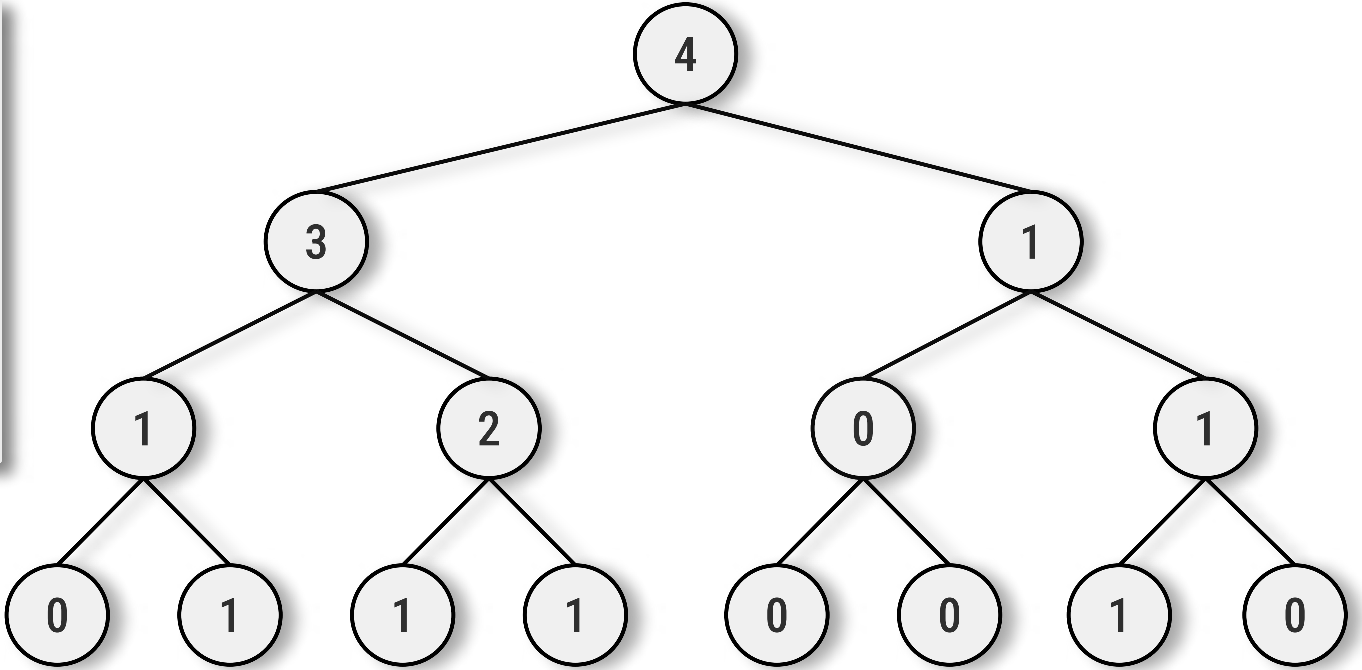


Properties of a Signal Tree

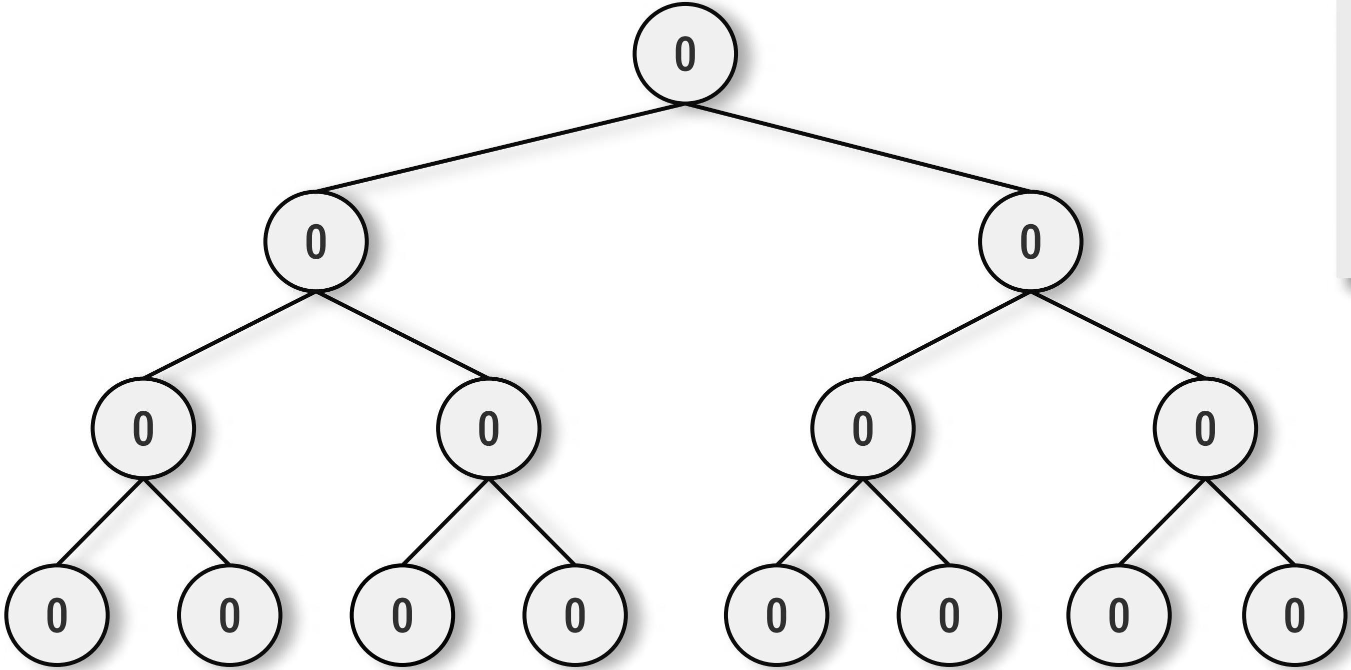
Perfect Binary Tree:
- Internal nodes have two children
- Leaf nodes all at the same level

Leaf Nodes Represent Signals:
- Zero indicates reset (off)
- One indicates set (on)

Child Sum Property:
- Internal nodes are sum of children



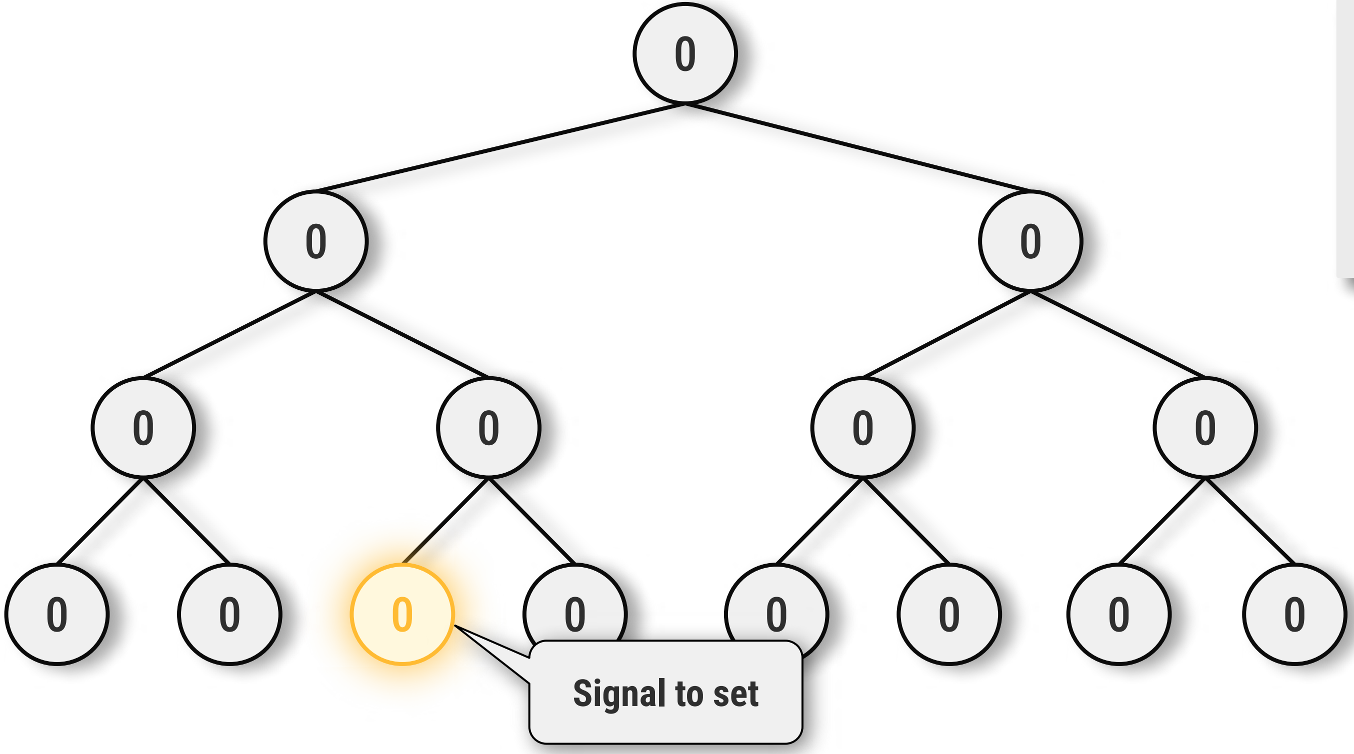
Setting a Signal



Order of updates:

- Begin at the leaf node
- Set node to one (on)
- Traverse tree upward from leaf to the root and increment each node by one

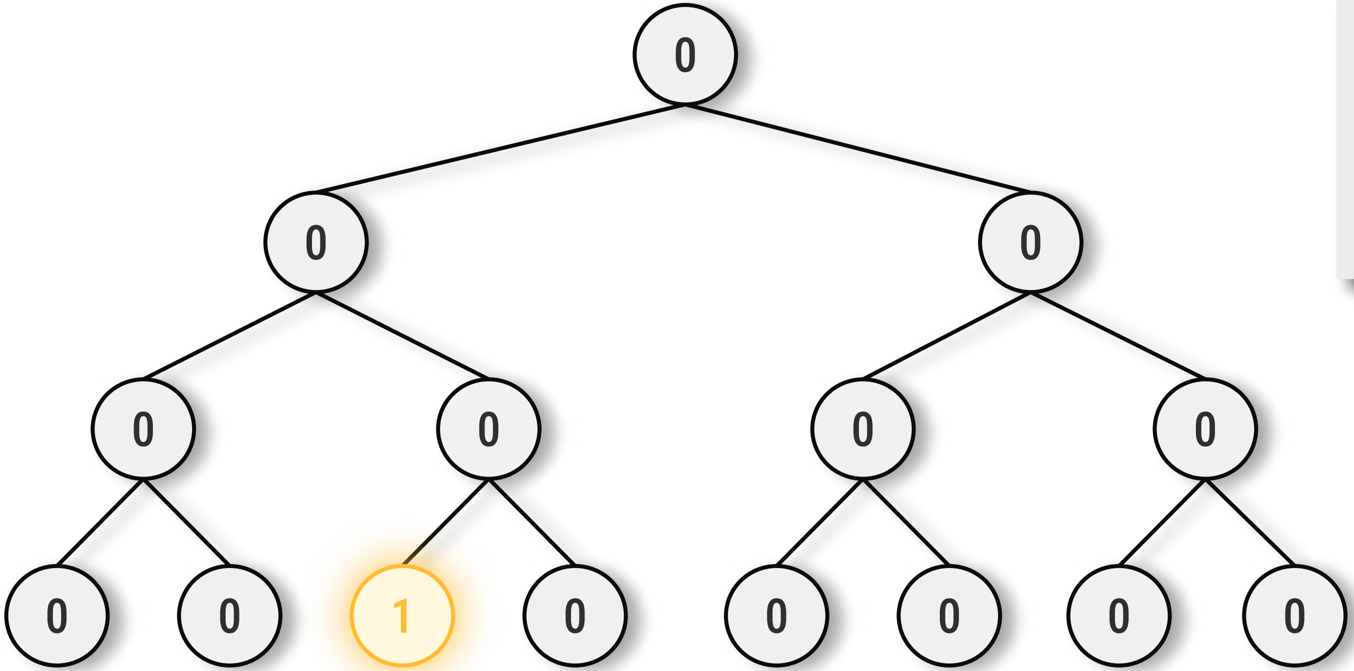
Setting a Signal



Order of updates:

- **Begin at the leaf node**
- Set node to one (on)
- Traverse tree upward from leaf to the root and increment each node by one

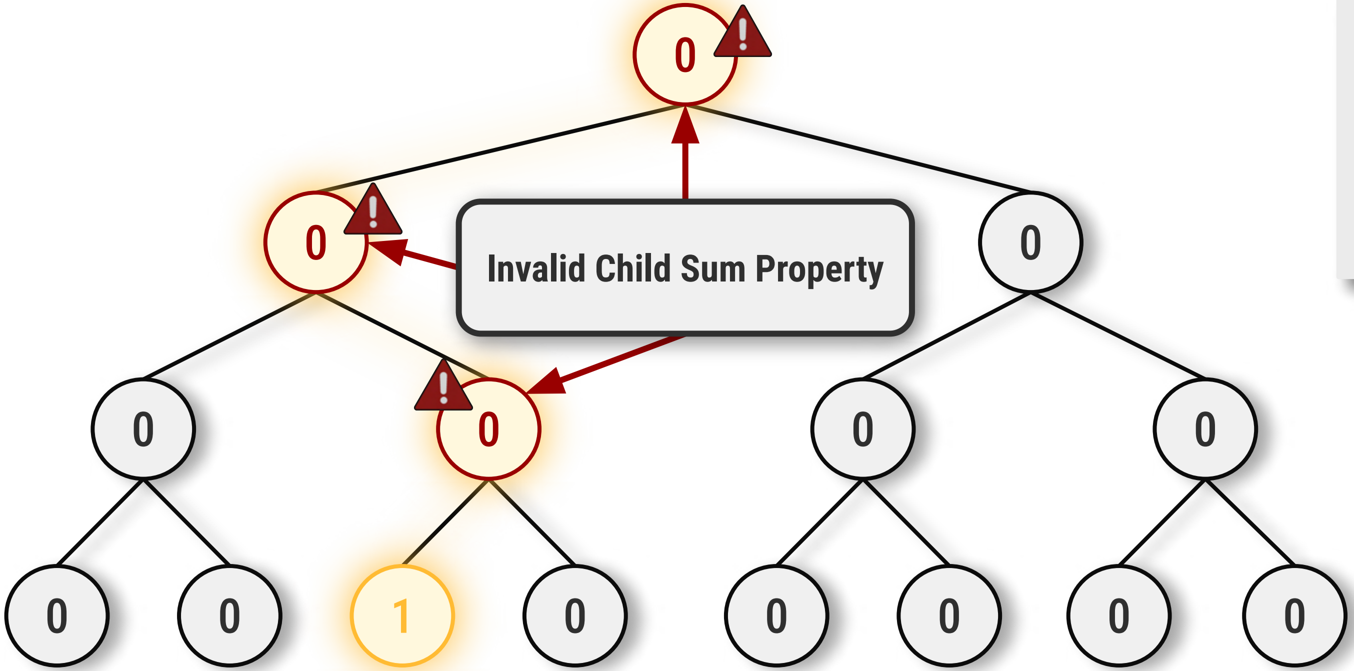
Setting a Signal



Order of updates:

- Begin at the leaf node
- Set node to one (on)
- Traverse tree upward from leaf to the root and increment each node by one

Setting a Signal

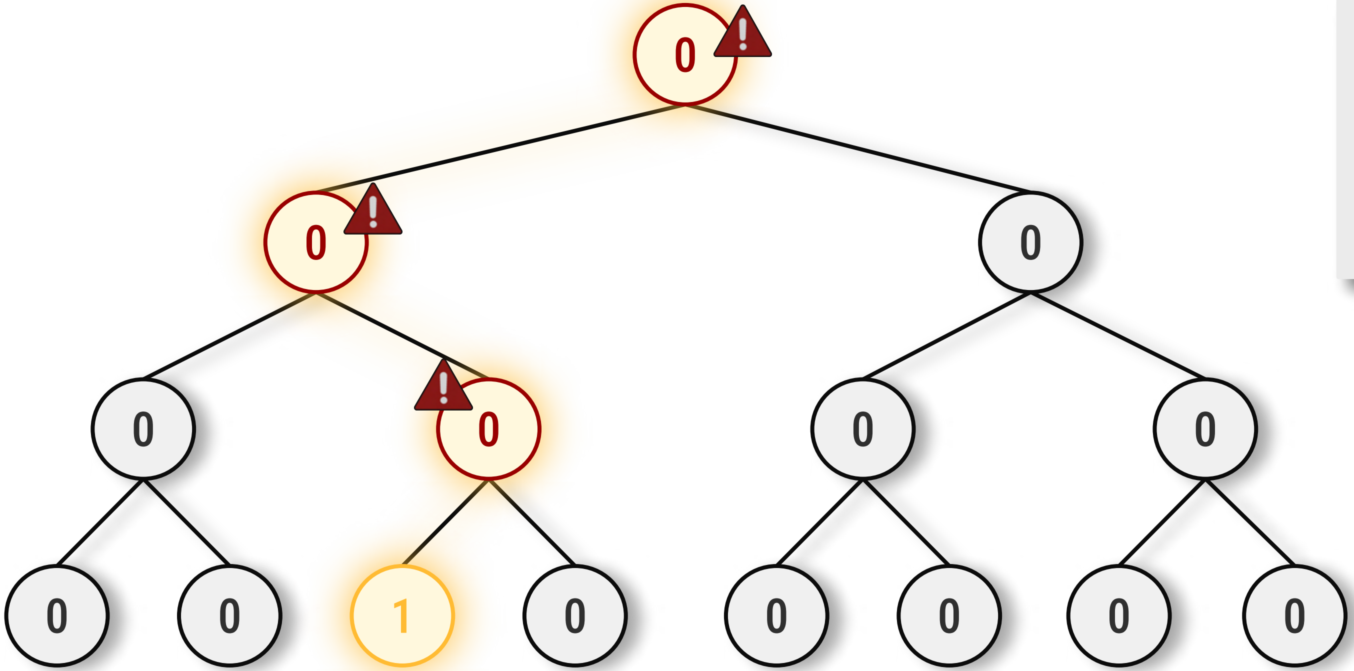


Invalid Child Sum Property

Order of updates:

- Begin at the leaf node
- Set node to one (on)
- Traverse tree upward from leaf to the root and increment each node by one

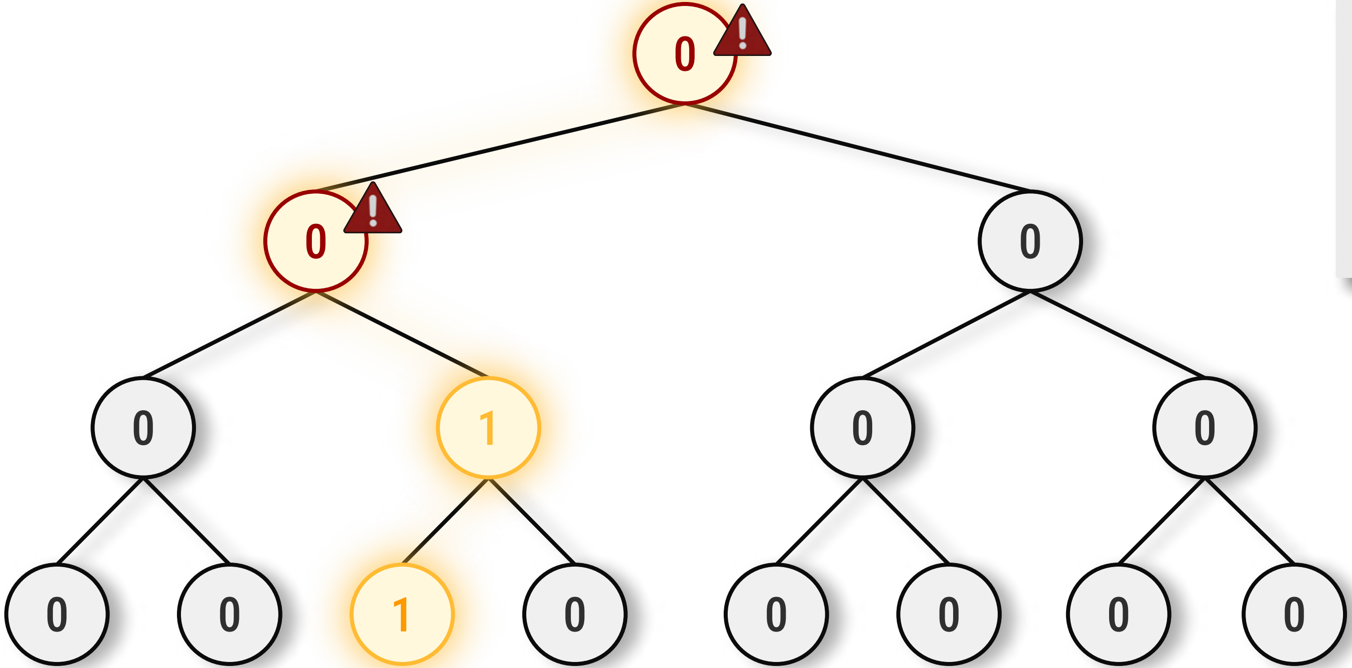
Setting a Signal



Order of updates:

- Begin at the leaf node
- Set node to one (on)
- Traverse tree upward from leaf to the root and increment each node by one

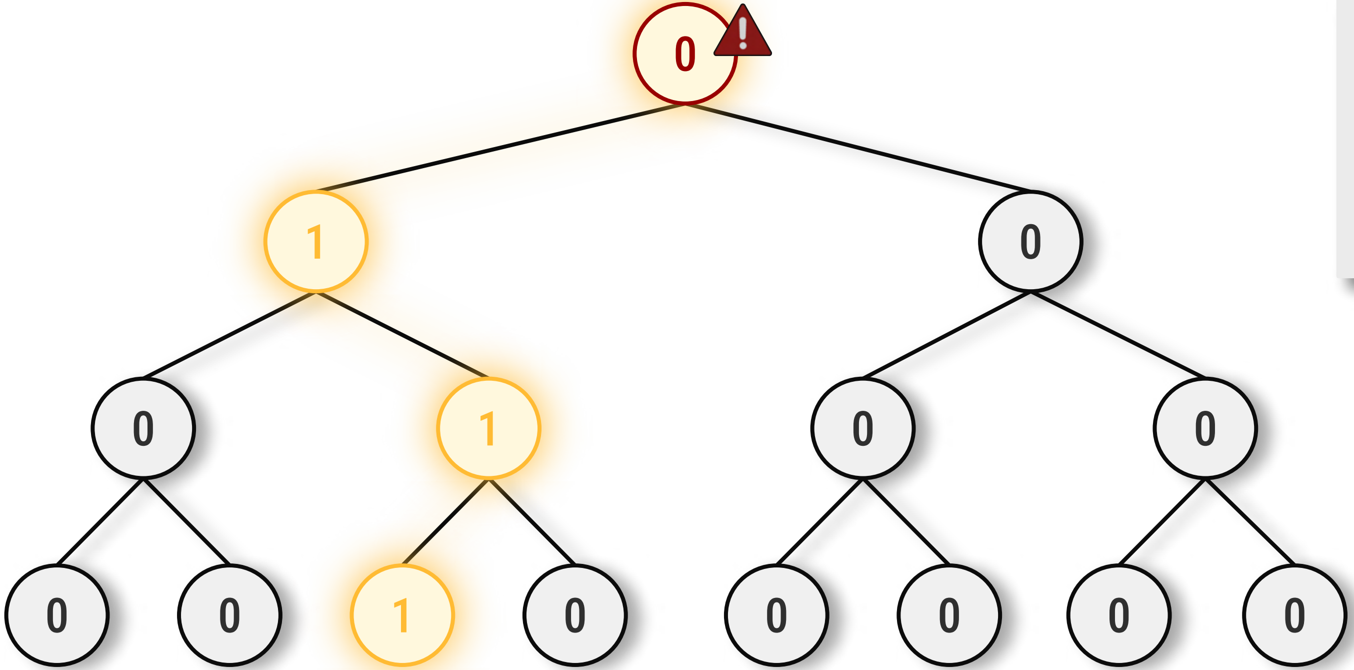
Setting a Signal



Order of updates:

- Begin at the leaf node
- Set node to one (on)
- **Traverse tree upward from leaf to the root and increment each node by one**

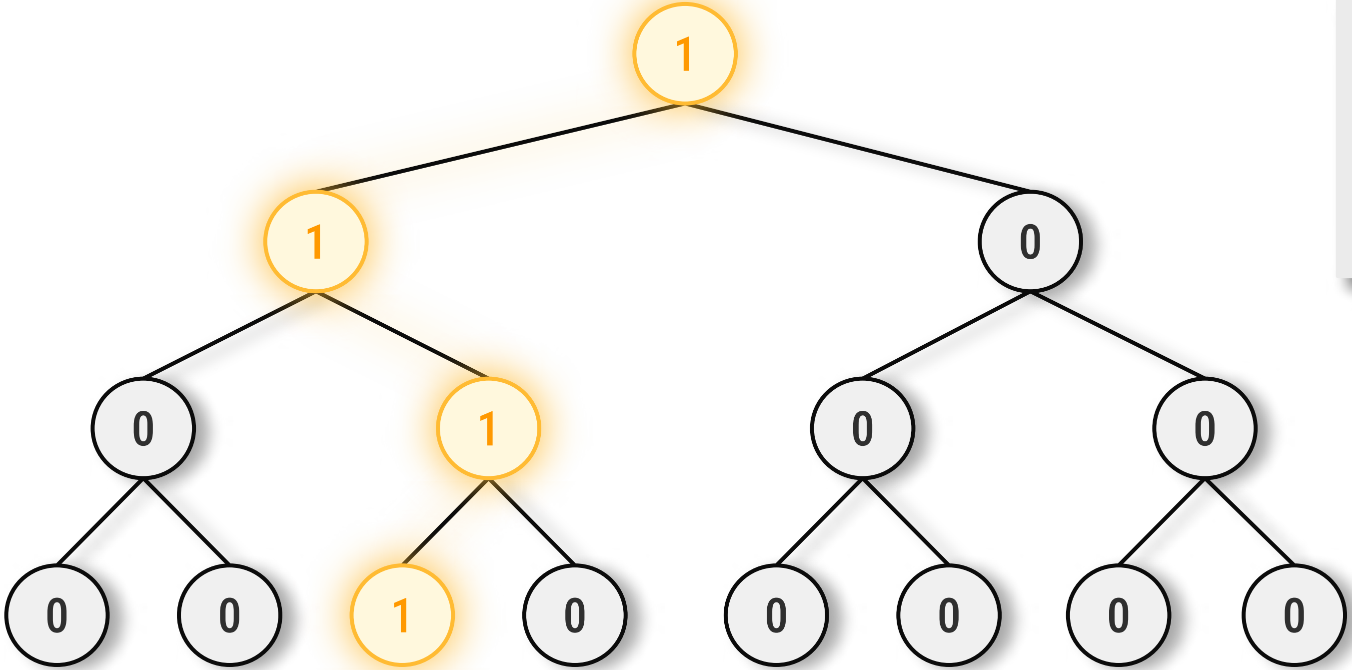
Setting a Signal



Order of updates:

- Begin at the leaf node
- Set node to one (on)
- Traverse tree upward from leaf to the root and increment each node by one

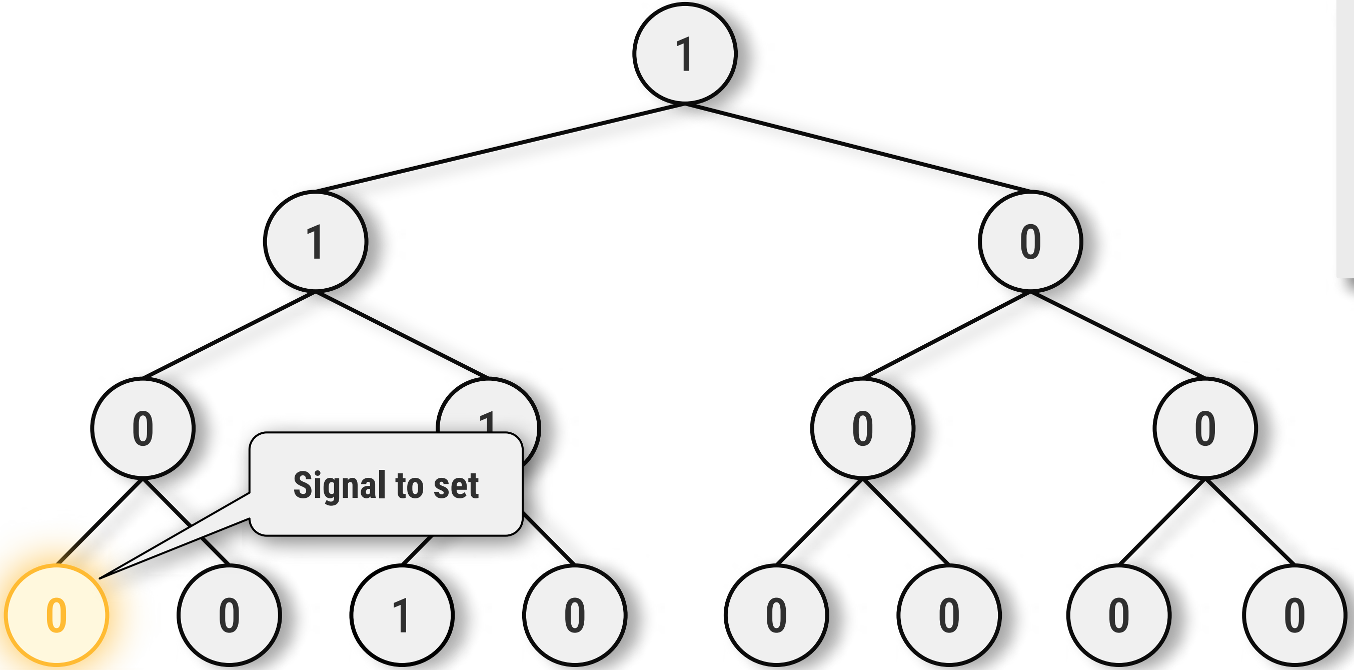
Setting a Signal



Order of updates:

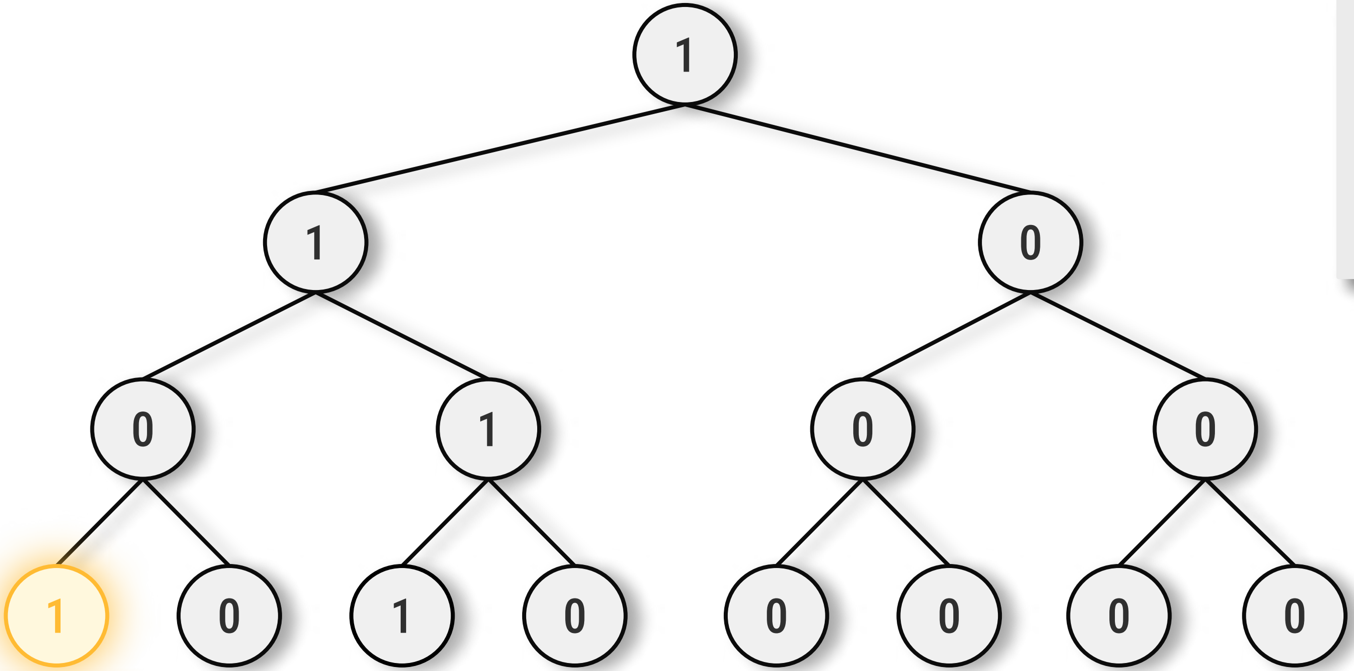
- Begin at the leaf node
- Set node to one (on)
- Traverse tree upward from leaf to the root and increment each node by one

Setting a Signal



Order of updates:
- **Begin at the leaf node**
- Set node to one (on)
- Traverse tree upward from leaf to the root and increment each node by one

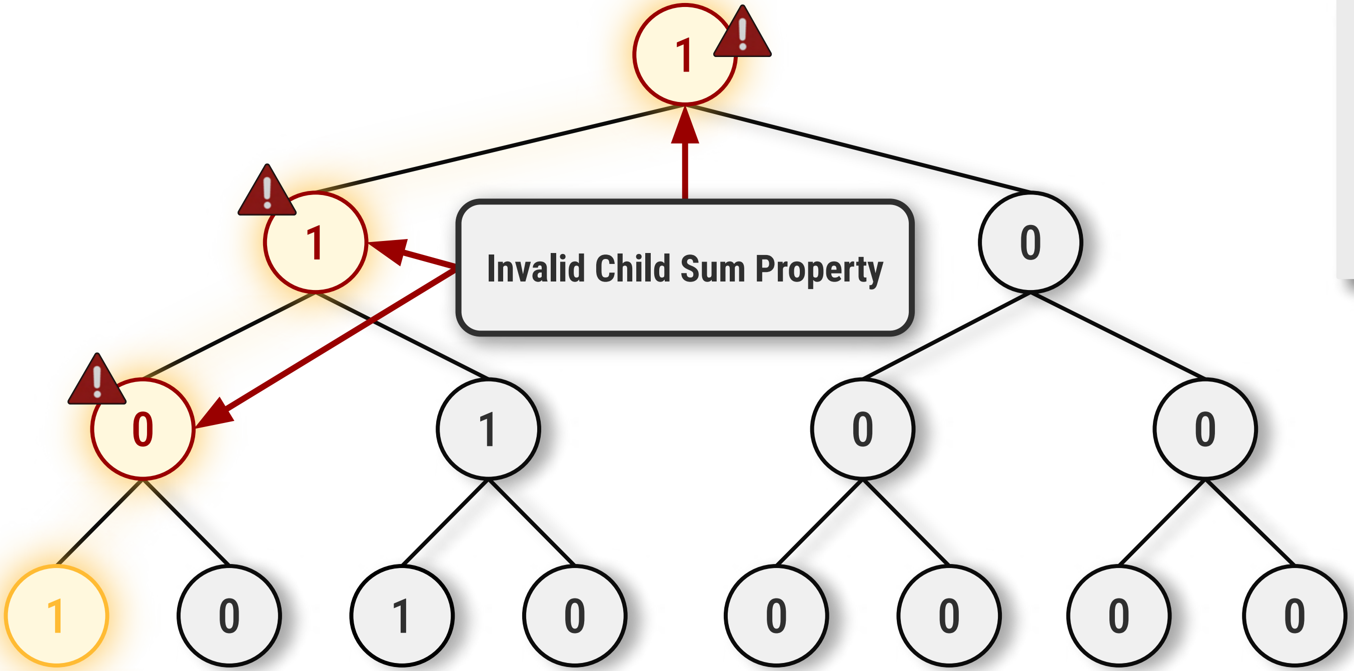
Setting a Signal



Order of updates:

- Begin at the leaf node
- Set node to one (on)
- Traverse tree upward from leaf to the root and increment each node by one

Setting a Signal

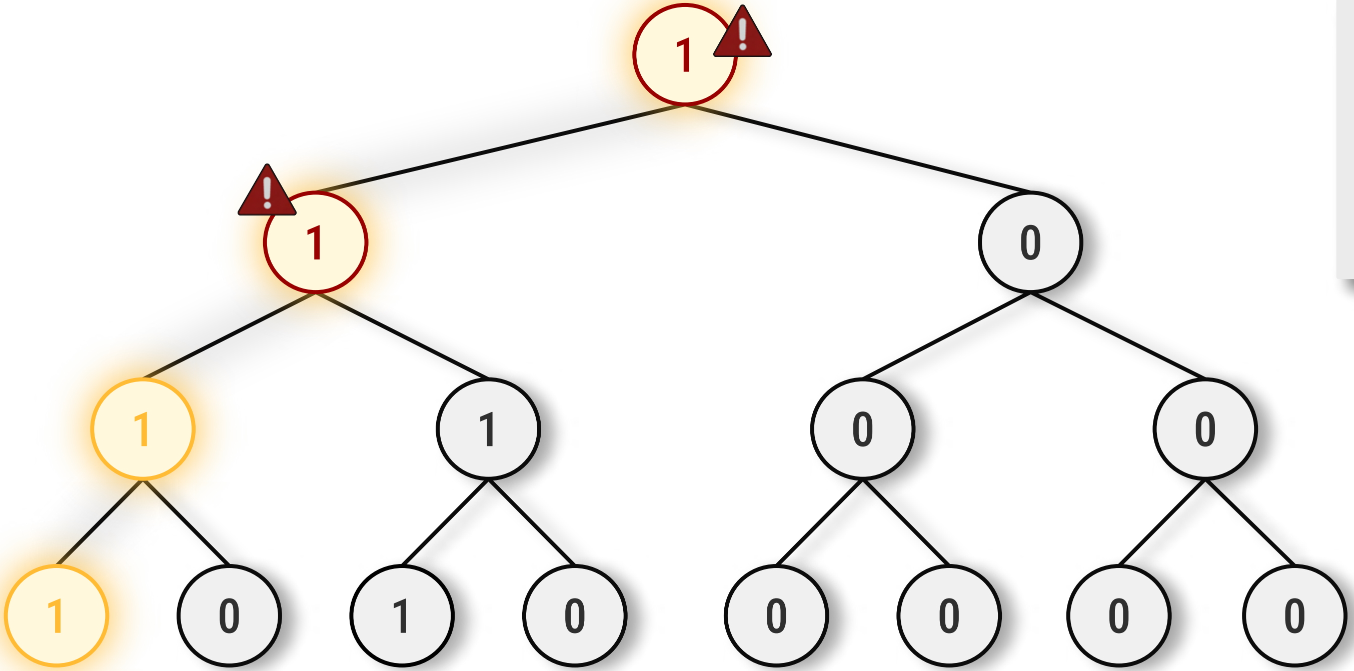


Invalid Child Sum Property

Order of updates:

- Begin at the leaf node
- Set node to one (on)
- Traverse tree upward from leaf to the root and increment each node by one

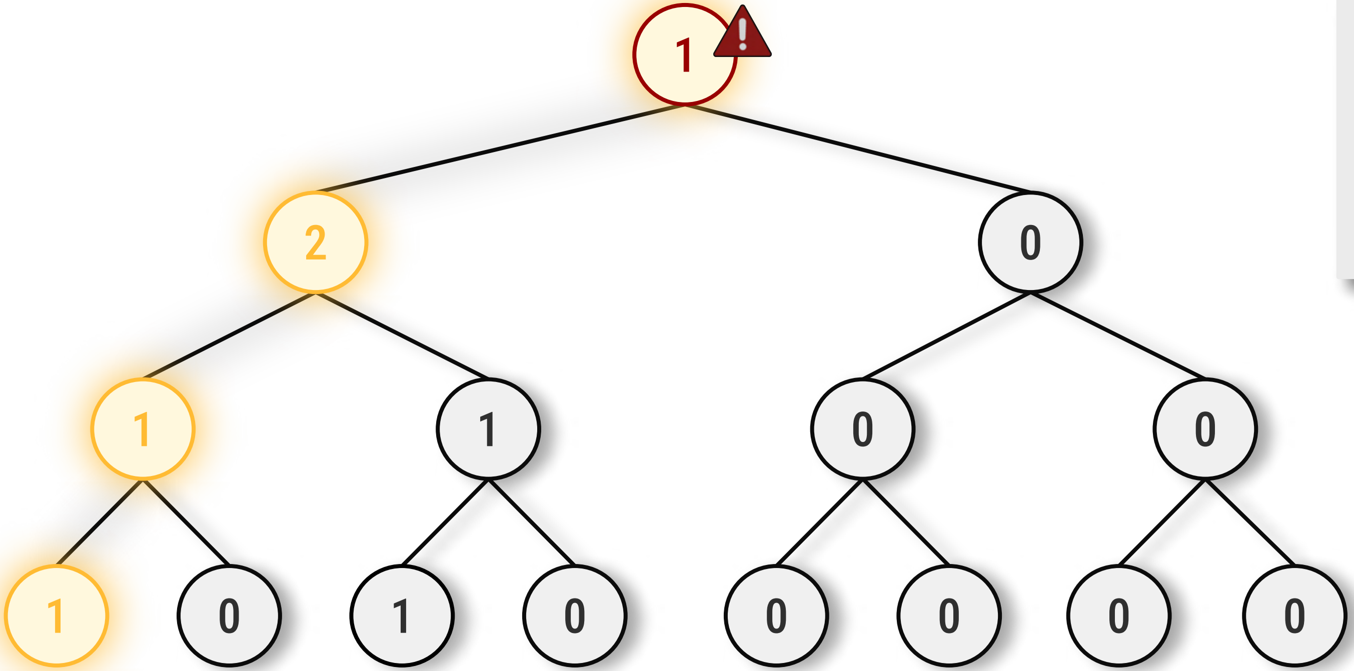
Setting a Signal



Order of updates:

- Begin at the leaf node
- Set node to one (on)
- Traverse tree upward from leaf to the root and increment each node by one

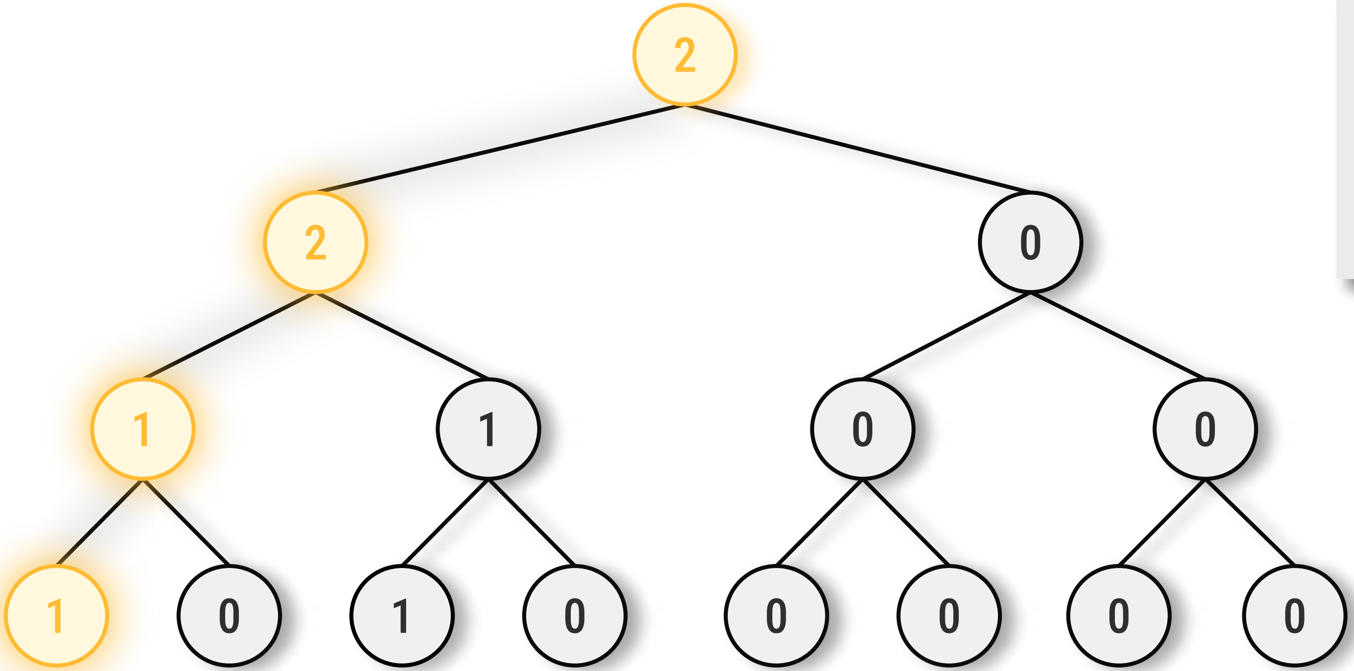
Setting a Signal



Order of updates:

- Begin at the leaf node
- Set node to one (on)
- Traverse tree upward from leaf to the root and increment each node by one

Setting a Signal



Order of updates:

- Begin at the leaf node
- Set node to one (on)
- Traverse tree upward from leaf to the root and increment each node by one

Setting Signals:

Summary

- Leaf nodes represent individual signals:
  - Zero indicates signal not set. One indicates signal is set.

- Internal nodes are atomic counters:
  - Count indicates the number of child leaf nodes and which are set to “on”
  - Determining if the tree contains any set signals is as simple as checking the root for zero

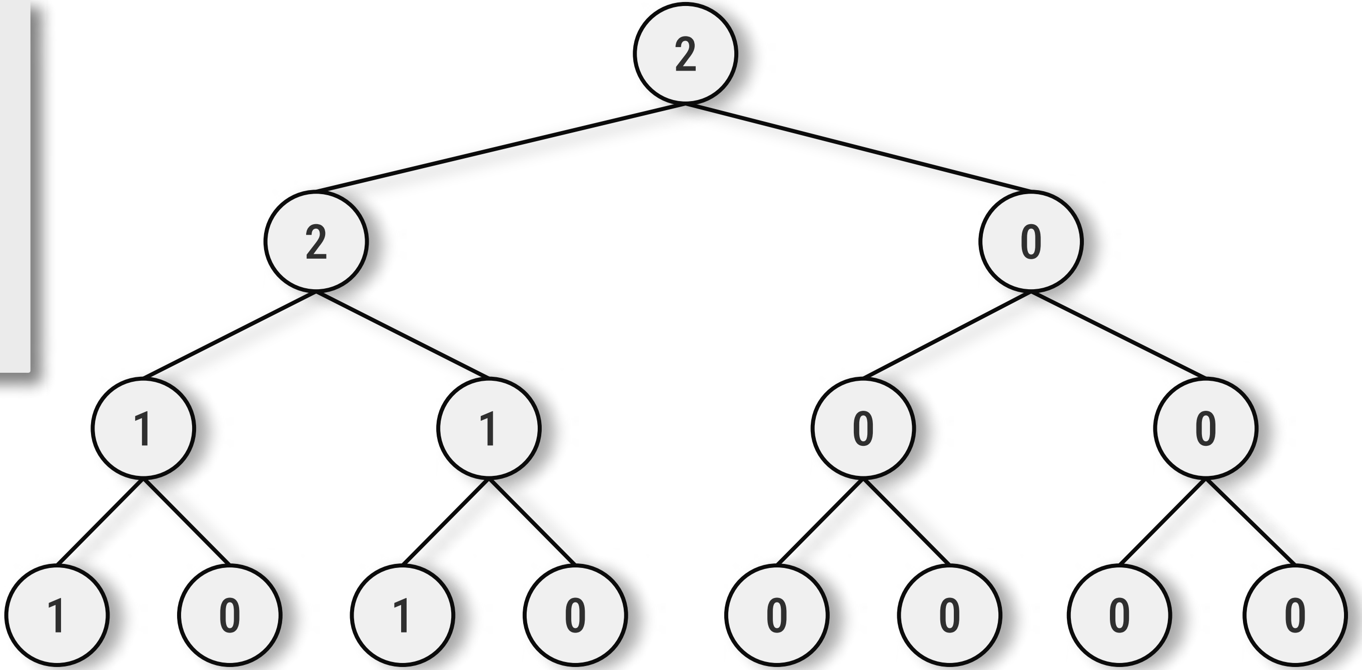
- Setting a signal is done by incrementing nodes along the path from leaf to root:
  - Order is important!

- All increments are done atomically:
  - Setting a signal is both lock free and wait free

Selecting Signals

Order of updates:

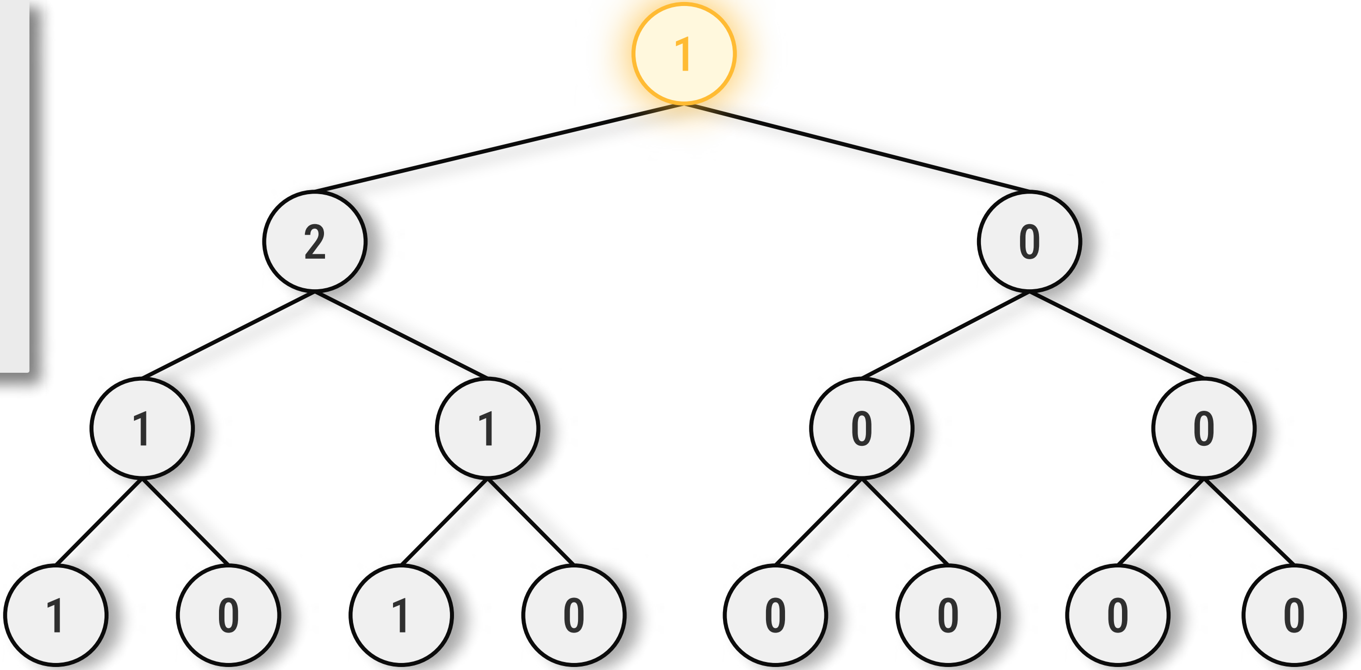
- Begin at the root node
- Decrement by one
- Traverse to a non-zero child node
- Decrement the child node
- Repeat until reaching and decrementing a leaf node from one to zero.



Selecting Signals

Order of updates:

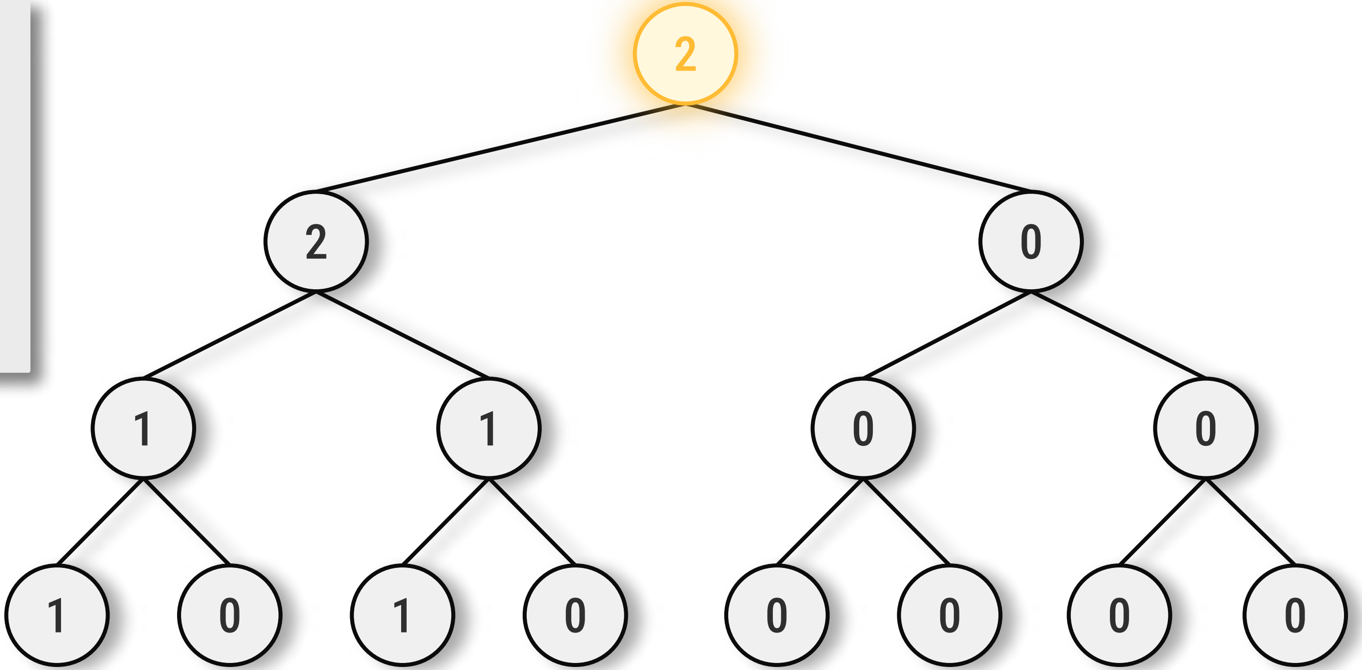
- Begin at the root node
- **Decrement by one**
- Traverse to a non-zero child node
- Decrement the child node
- Repeat until reaching and decrementing a leaf node from one to zero.



Selecting Signals

Order of updates:

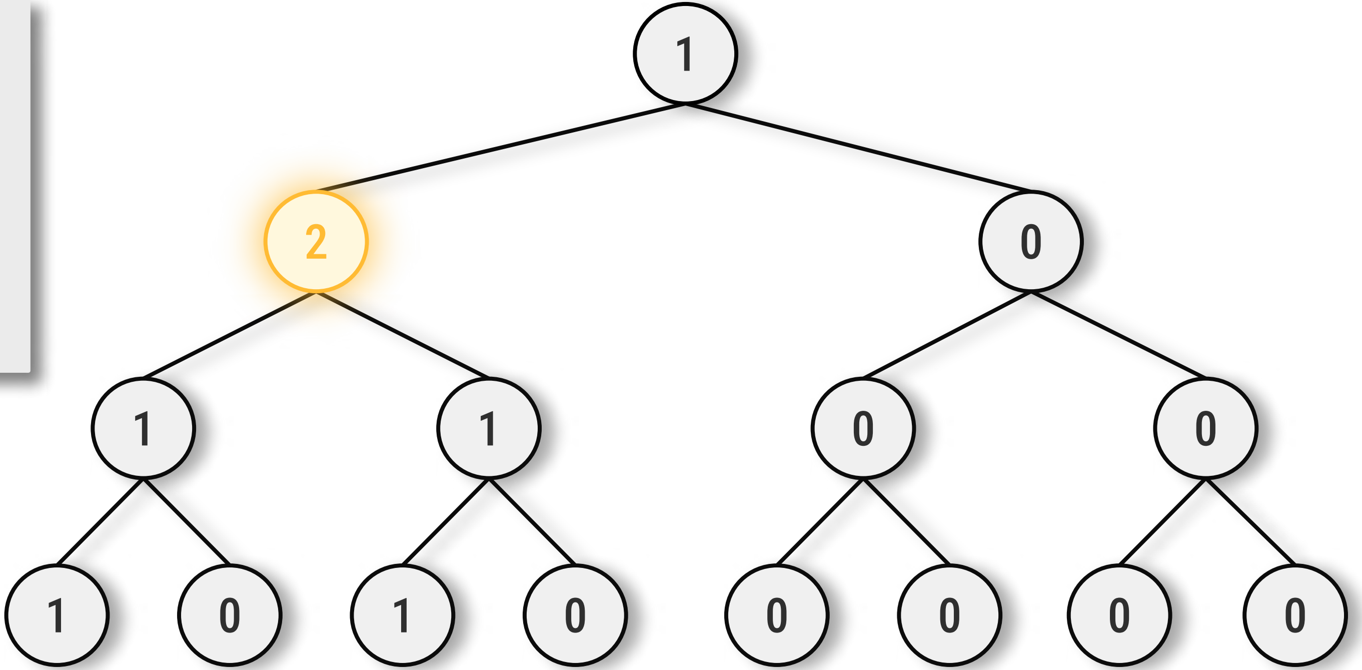
- **Begin at the root node**
- Decrement by one
- Traverse to a non-zero child node
- Decrement the child node
- Repeat until reaching and decrementing a leaf node from one to zero.



Selecting Signals

Order of updates:

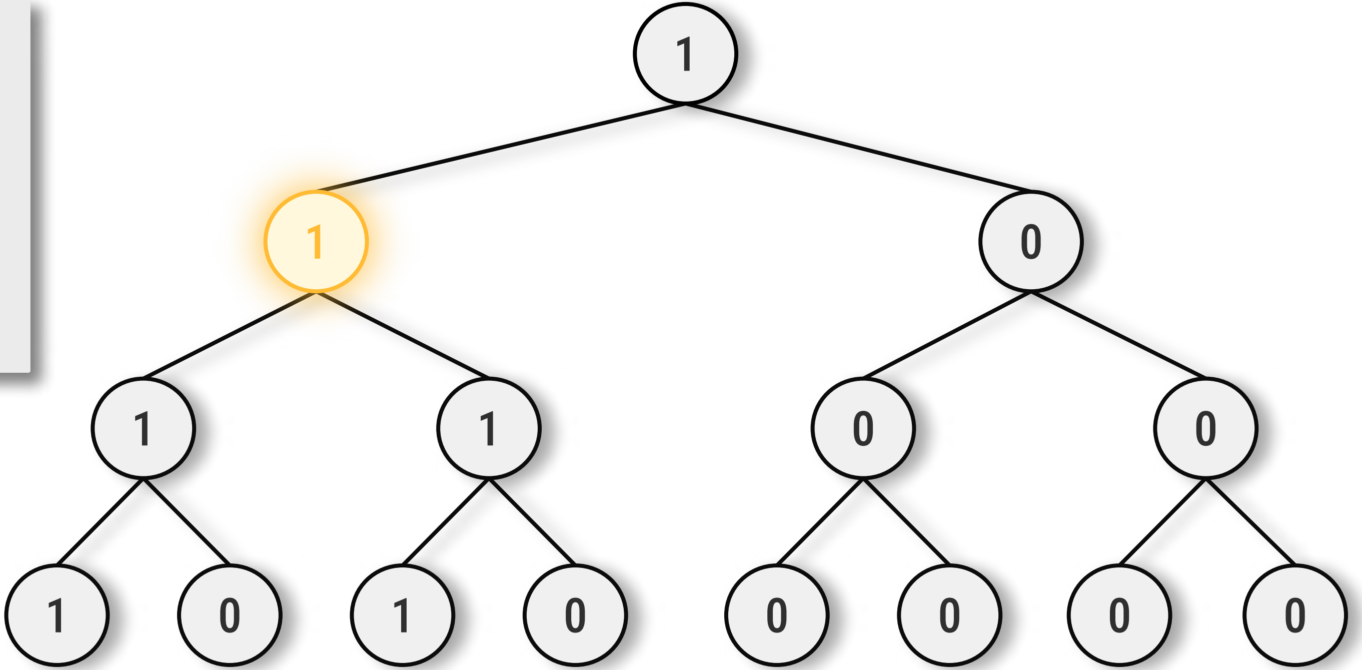
- Begin at the root node
- Decrement by one
- **Traverse to a non-zero child node**
- Decrement the child node
- Repeat until reaching and decrementing a leaf node from one to zero.



Selecting Signals

Order of updates:

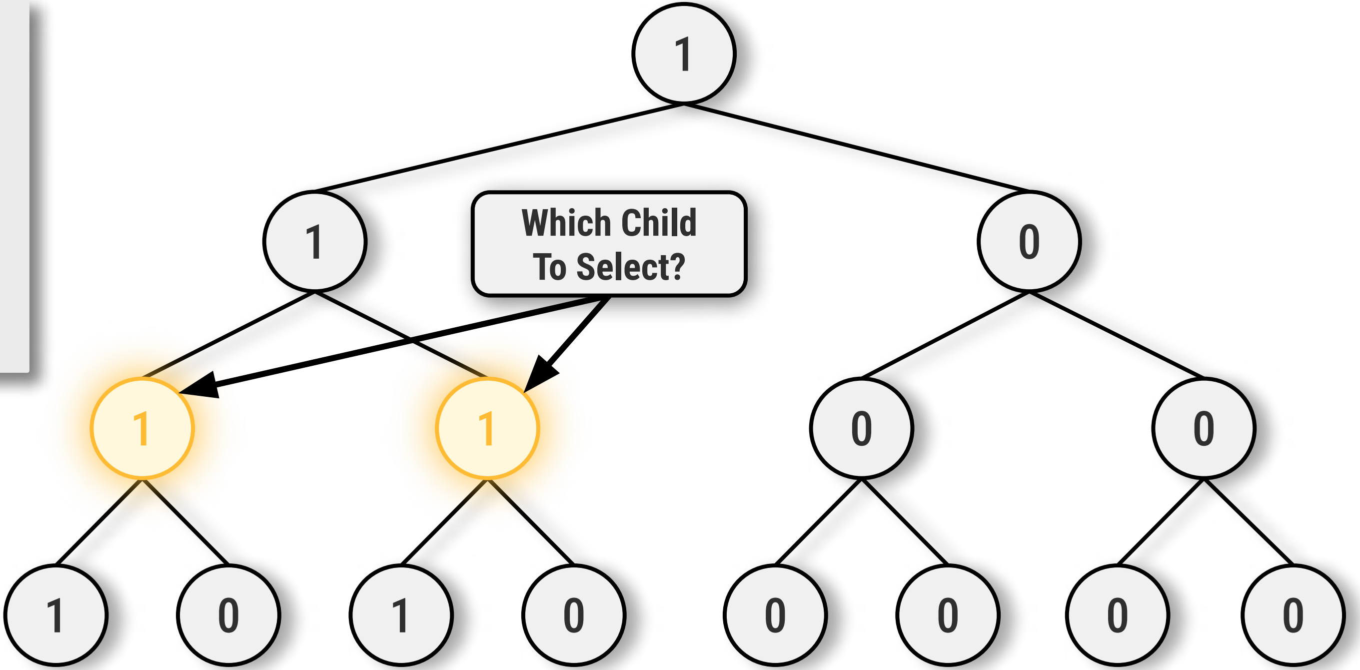
- Begin at the root node
- Decrement by one
- Traverse to a non-zero child node
- **Decrement the child node**
- Repeat until reaching and decrementing a leaf node from one to zero.



Selecting Signals

Order of updates:

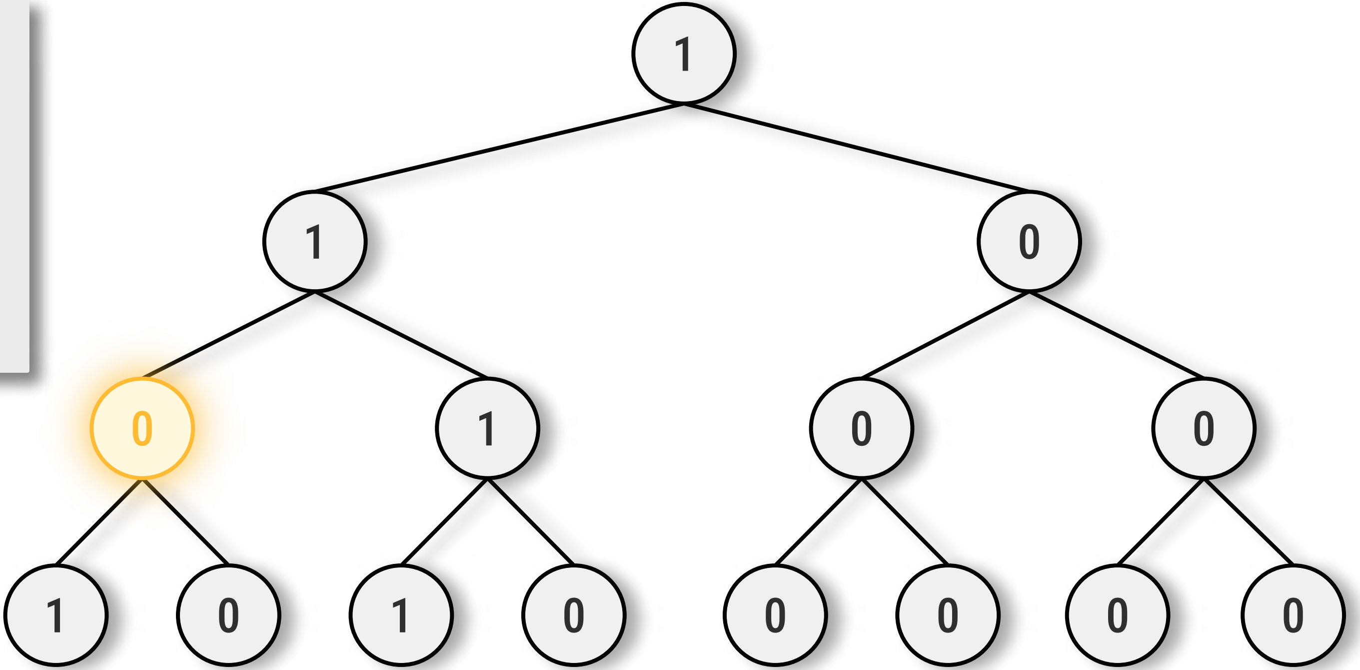
- Begin at the root node
- Decrement by one
- Traverse to a non-zero child node
- Decrement the child node
- **Repeat until reaching and decrementing a leaf node from one to zero.**



Selecting Signals

Order of updates:

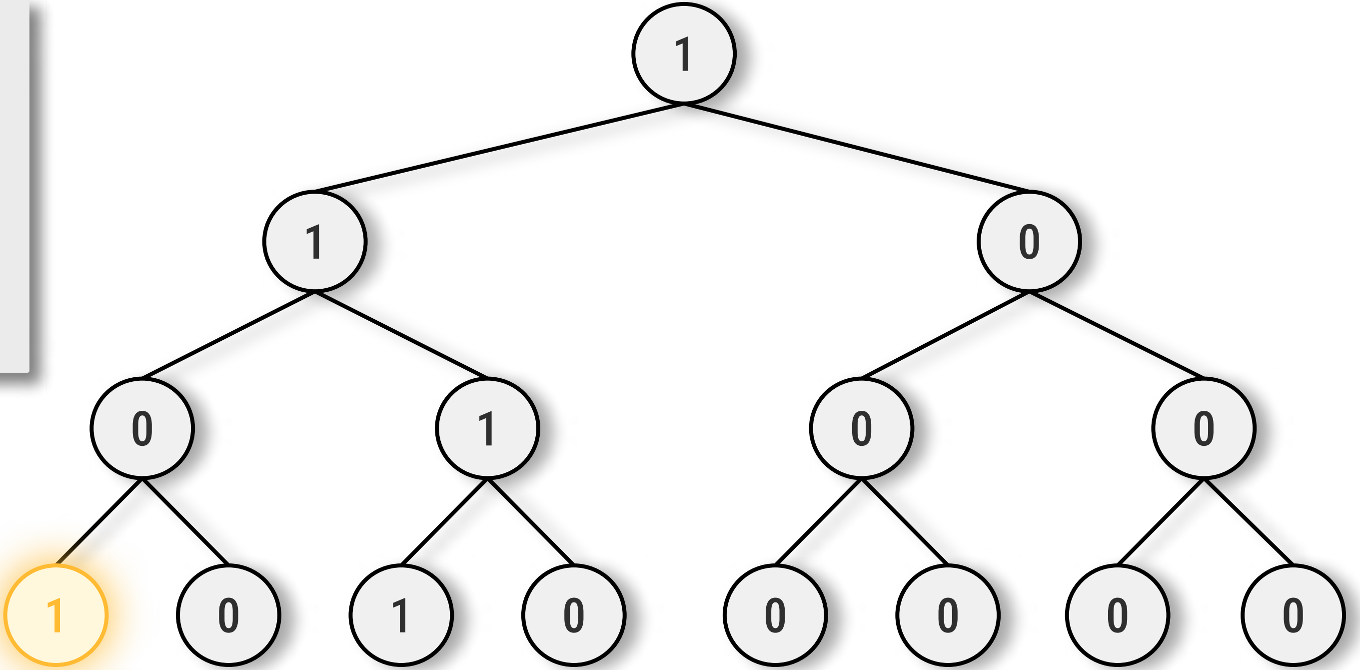
- Begin at the root node
- Decrement by one
- Traverse to a non-zero child node
- **Decrement the child node**
- Repeat until reaching and decrementing a leaf node from one to zero.



Selecting Signals

Order of updates:

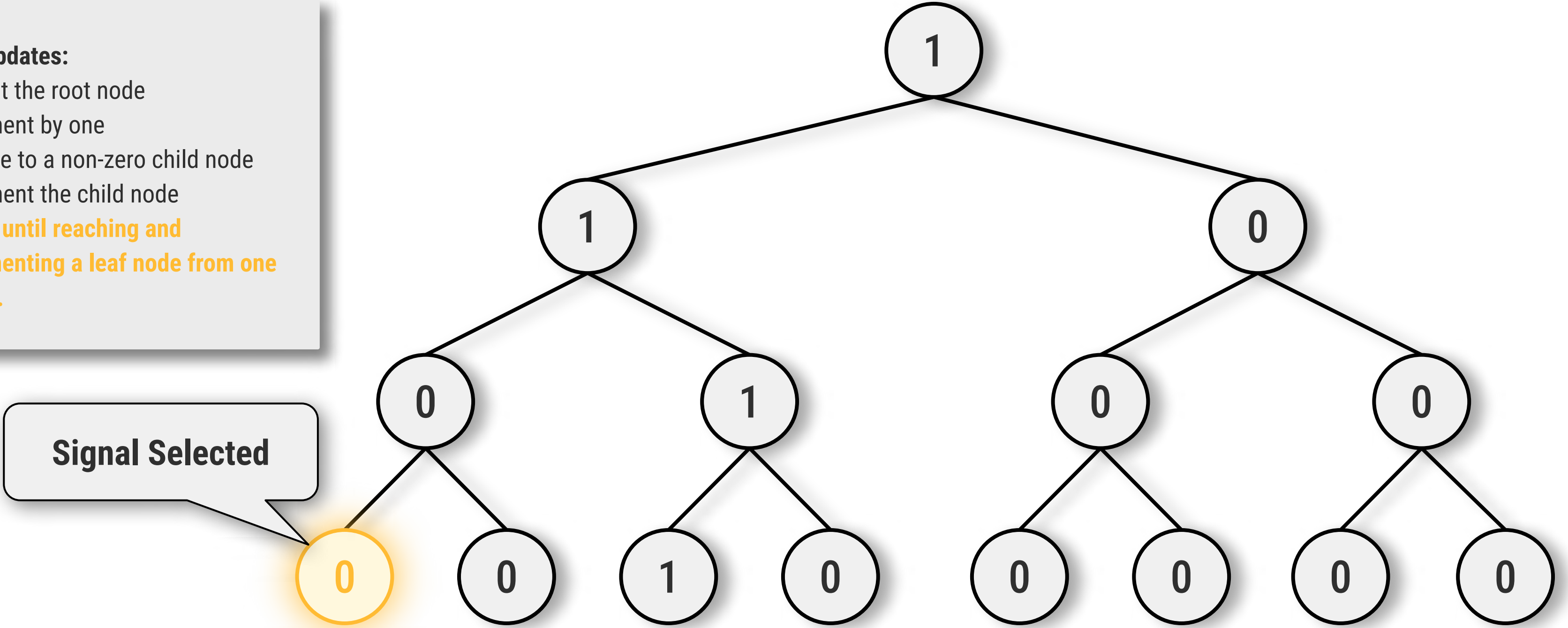
- Begin at the root node
- Decrement by one
- **Traverse to a non-zero child node**
- Decrement the child node
- Repeat until reaching and decrementing a leaf node from one to zero.



Selecting Signals

Order of updates:

- Begin at the root node
- Decrement by one
- Traverse to a non-zero child node
- Decrement the child node
- Repeat **until reaching and decrementing a leaf node from one to zero.**



Selecting Signals:

Summary

- Selecting signals is done from root to leaf
    - Order is important!

- All decrements are done atomically:
    - Setting selection is lock free

- Signals are located by following a path of non-zero nodes
    - When more than one possible path exists bias can be introduced to influence selection for either ‘fairness’ or for ‘prioritization’ of task selection

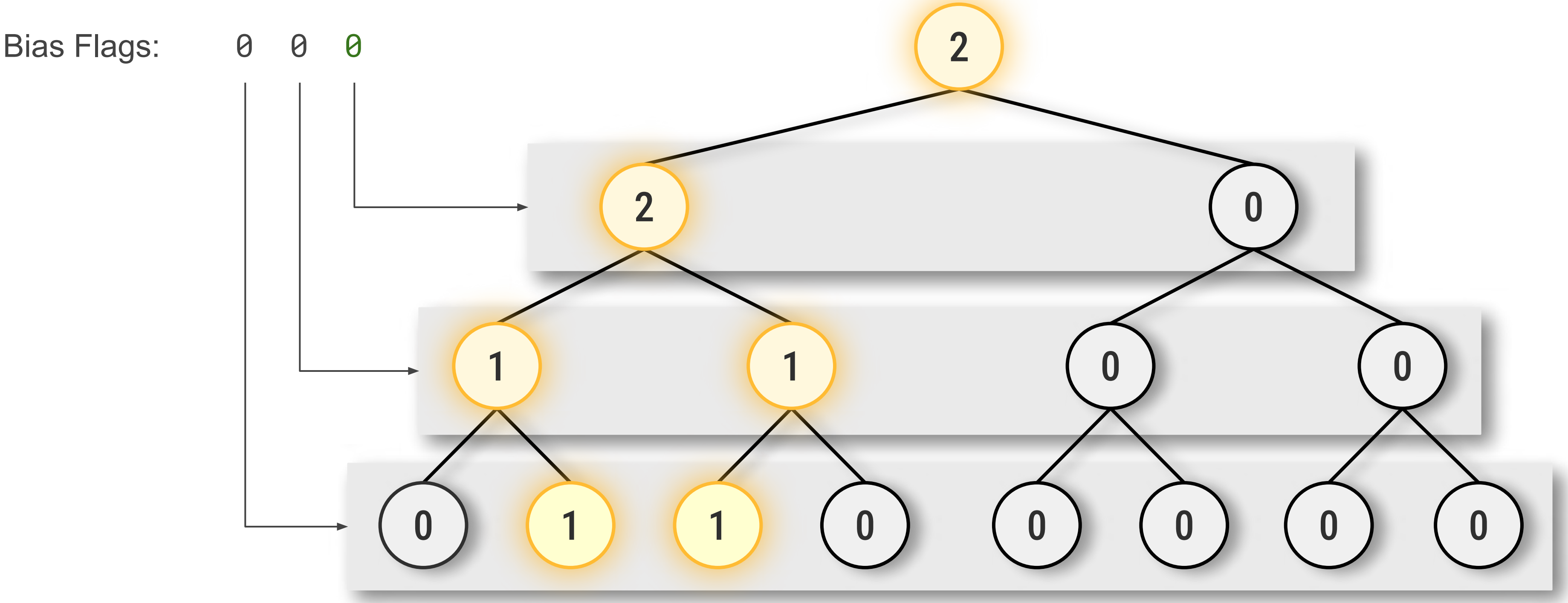
Creating Bias

Task Fairness And Prioritization

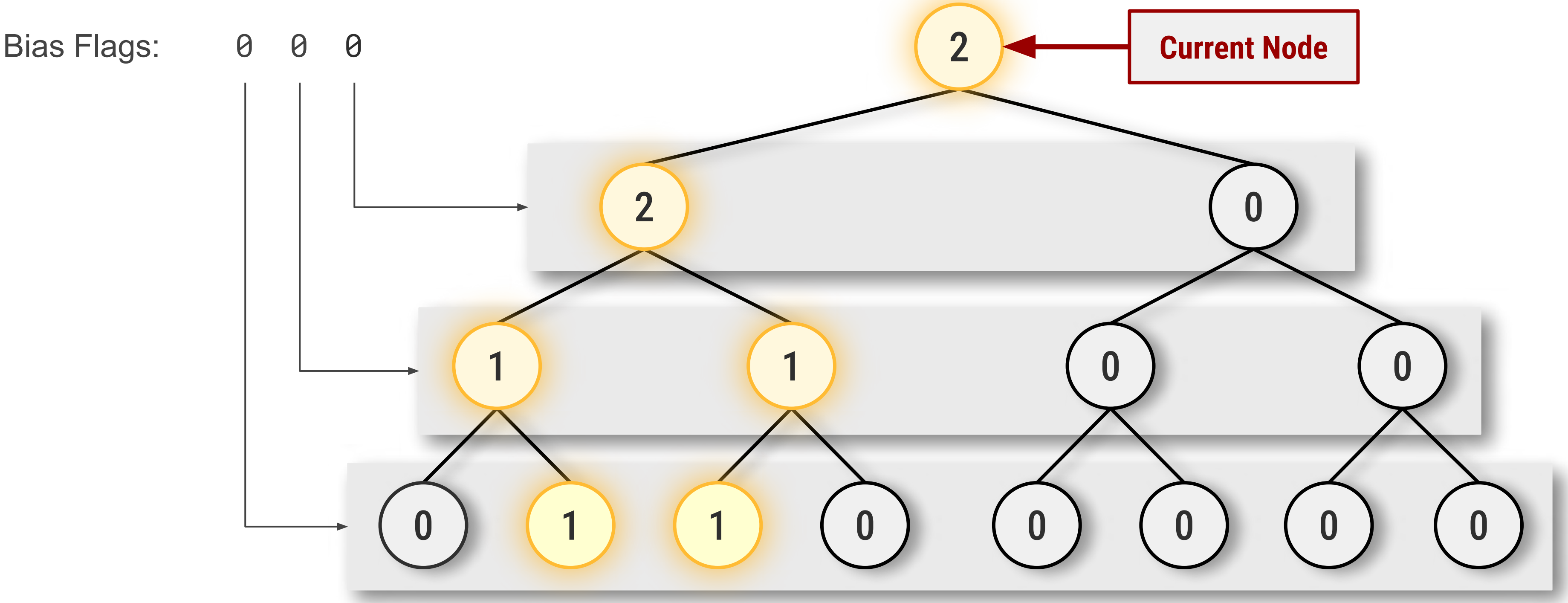
Bias Flags:

- An integer counter is used, where each bit corresponds to a unique level of the tree
- During signal selection (traversal from root to leaf) if the bit associated with the current tree level is:
  - If bit is zero then select left child, if possible
  - If bit is one then select right child, if possible
- This integer field is adjusted prior to each tree traversal:
  - For ‘fairness’ the counter is monotonically incremented *
  - To create ‘priority’ the bits which correspond to the top tree levels can be fixed to favor traversal to a particular branch of the overall tree IFF there are signals set in that section of the tree.
- Selection bias is per thread rather than per tree - unlike queues

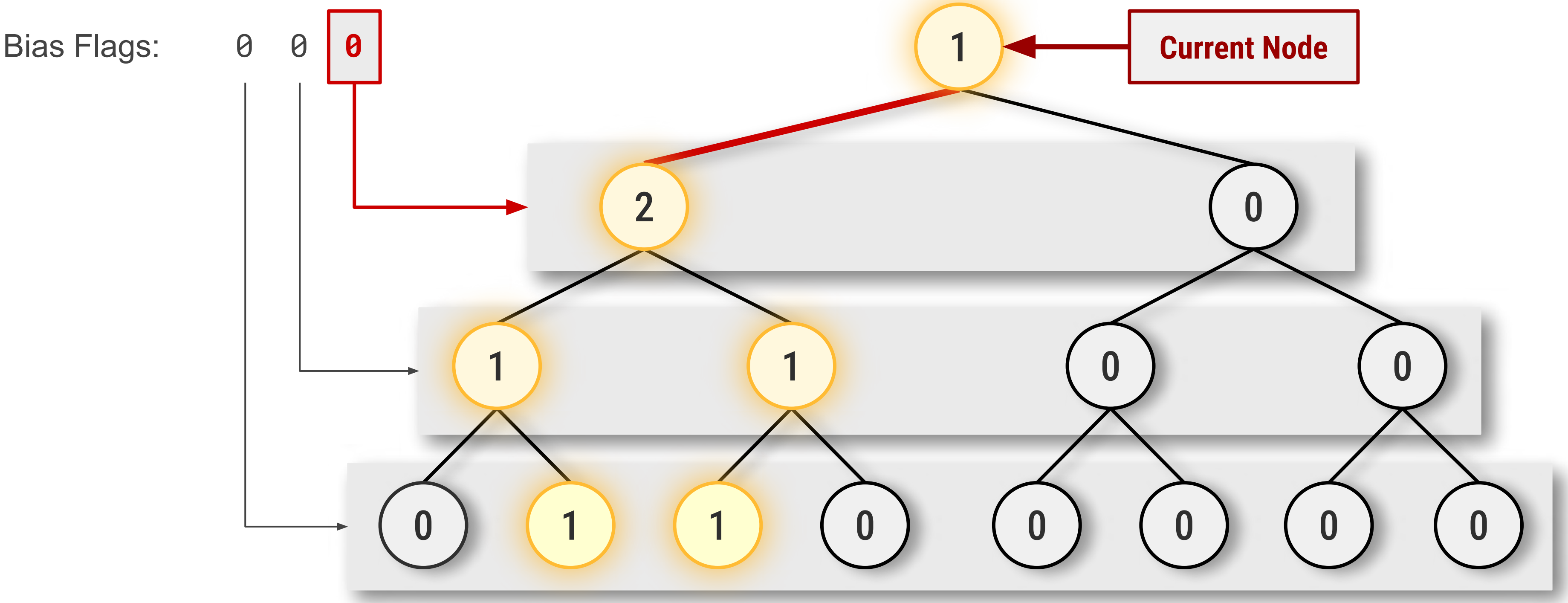
Bias: Fairness



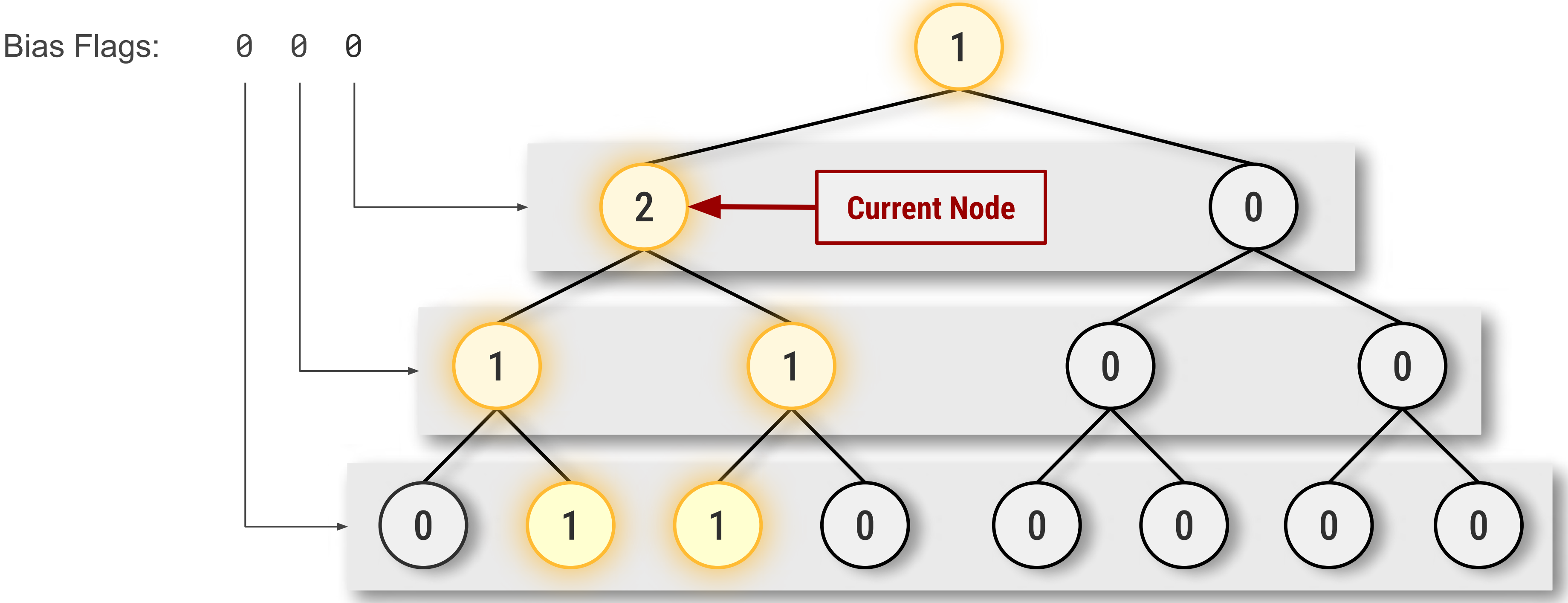
Bias: Fairness



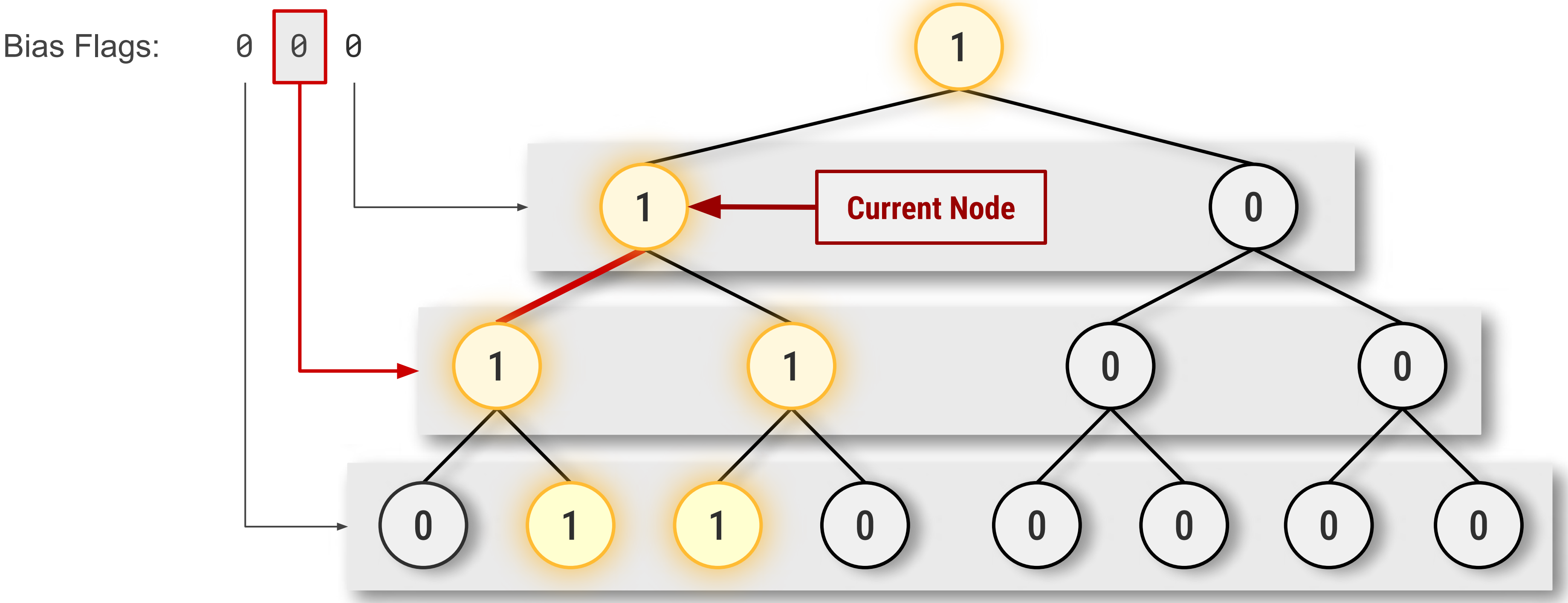
Bias: Fairness



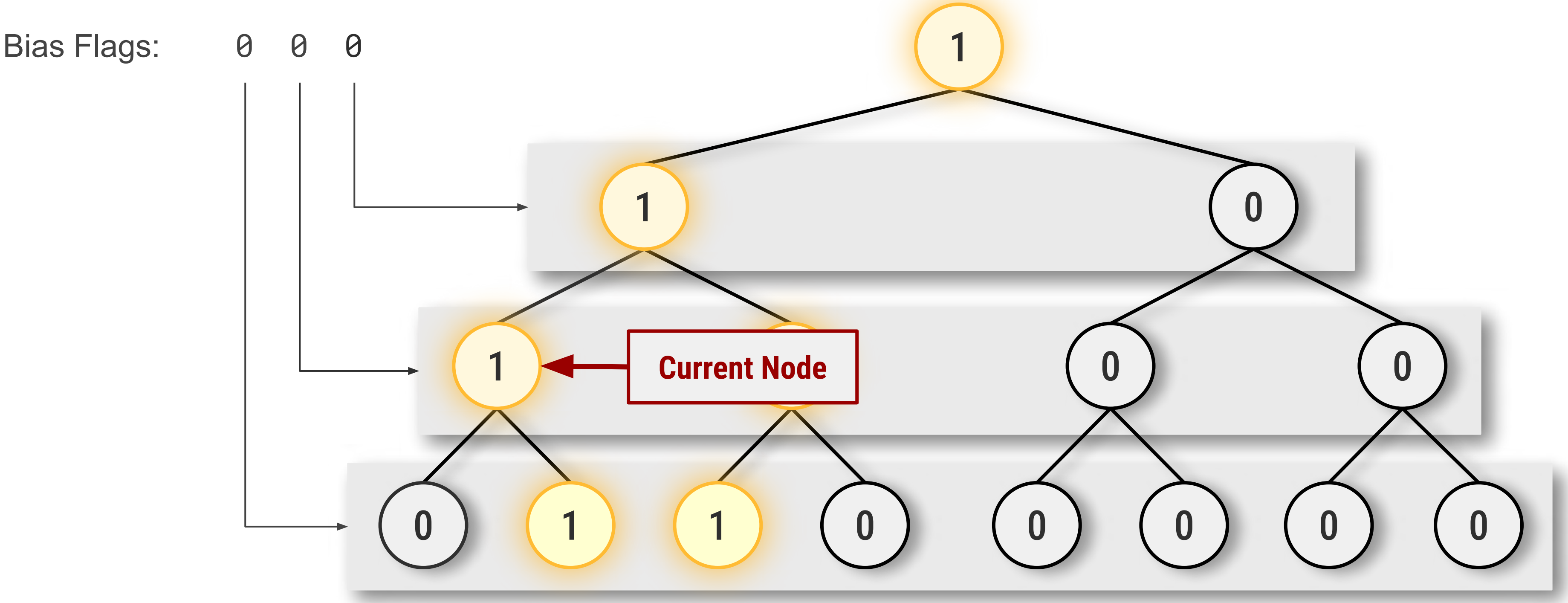
Bias: Fairness



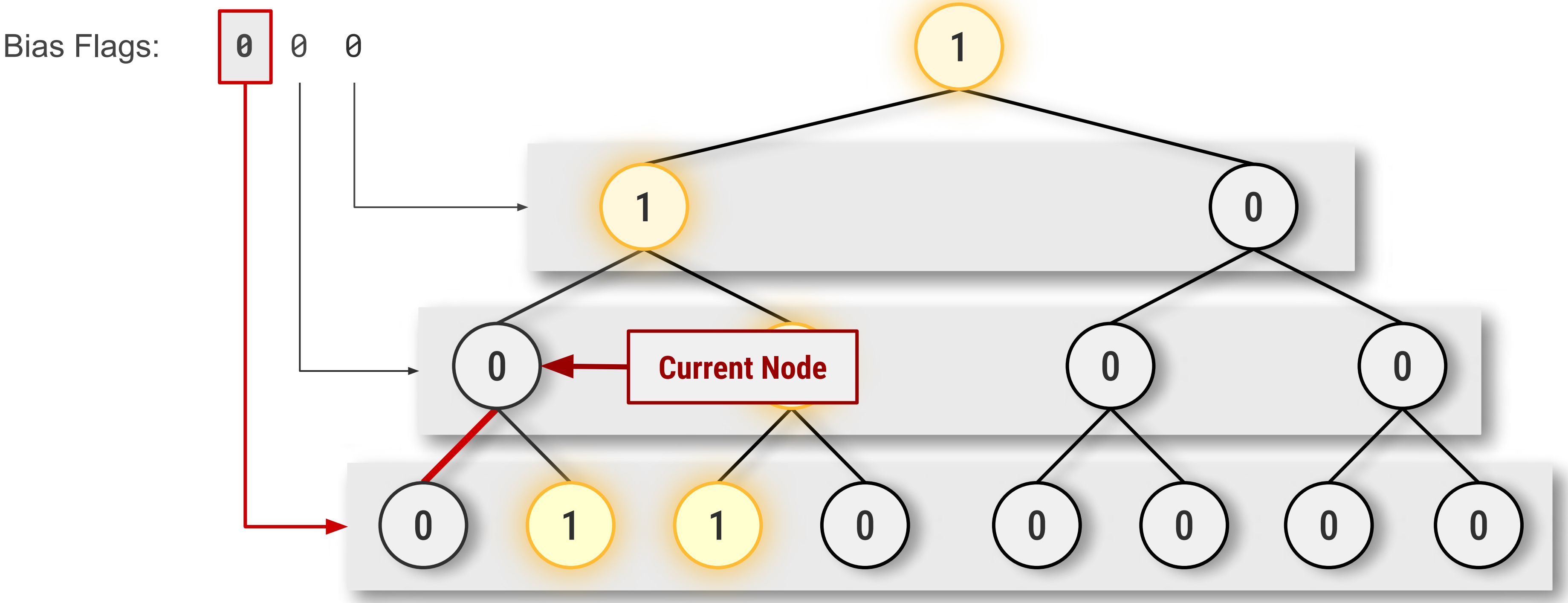
Bias: Fairness



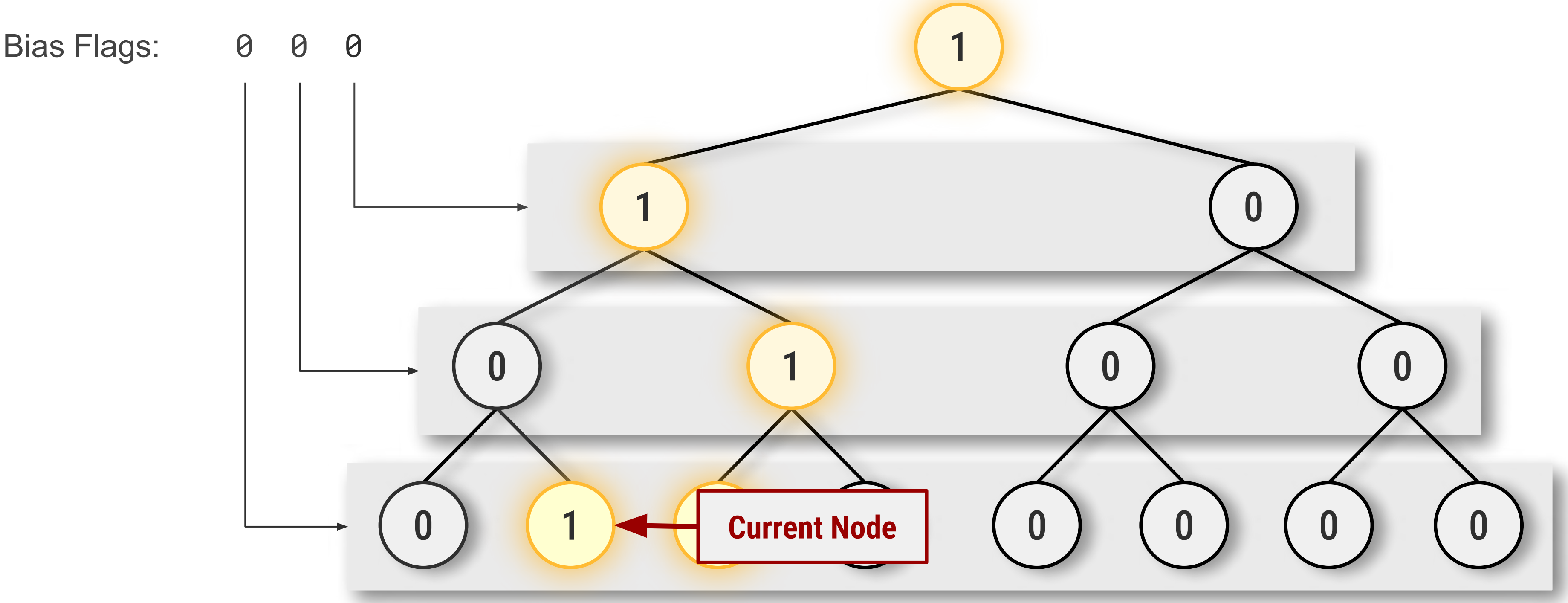
Bias: Fairness



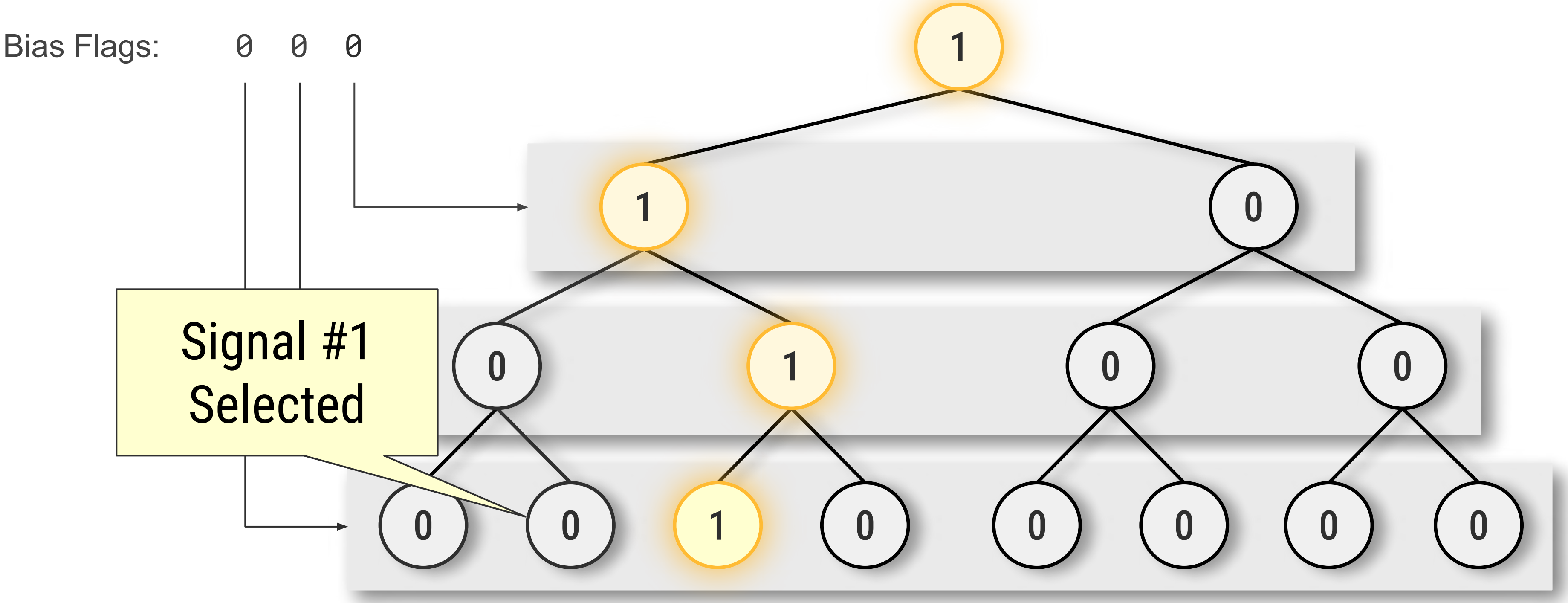
Bias: Fairness



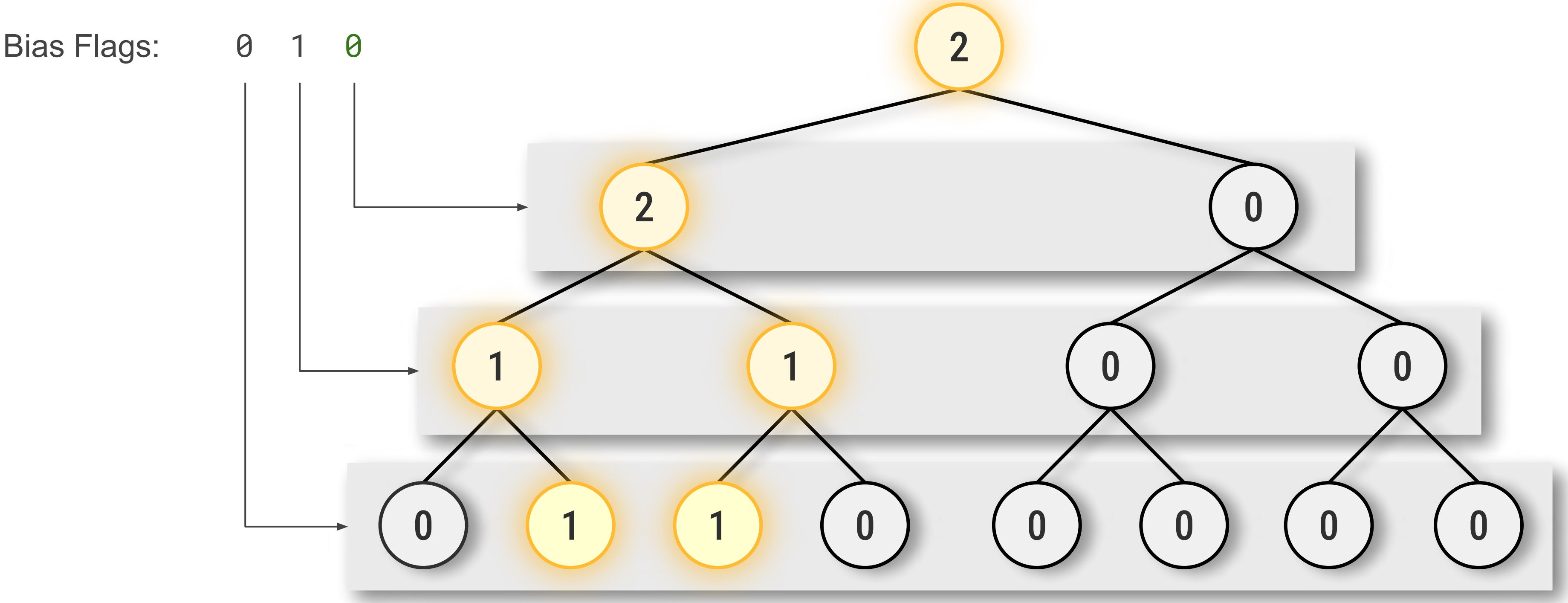
Bias: Fairness



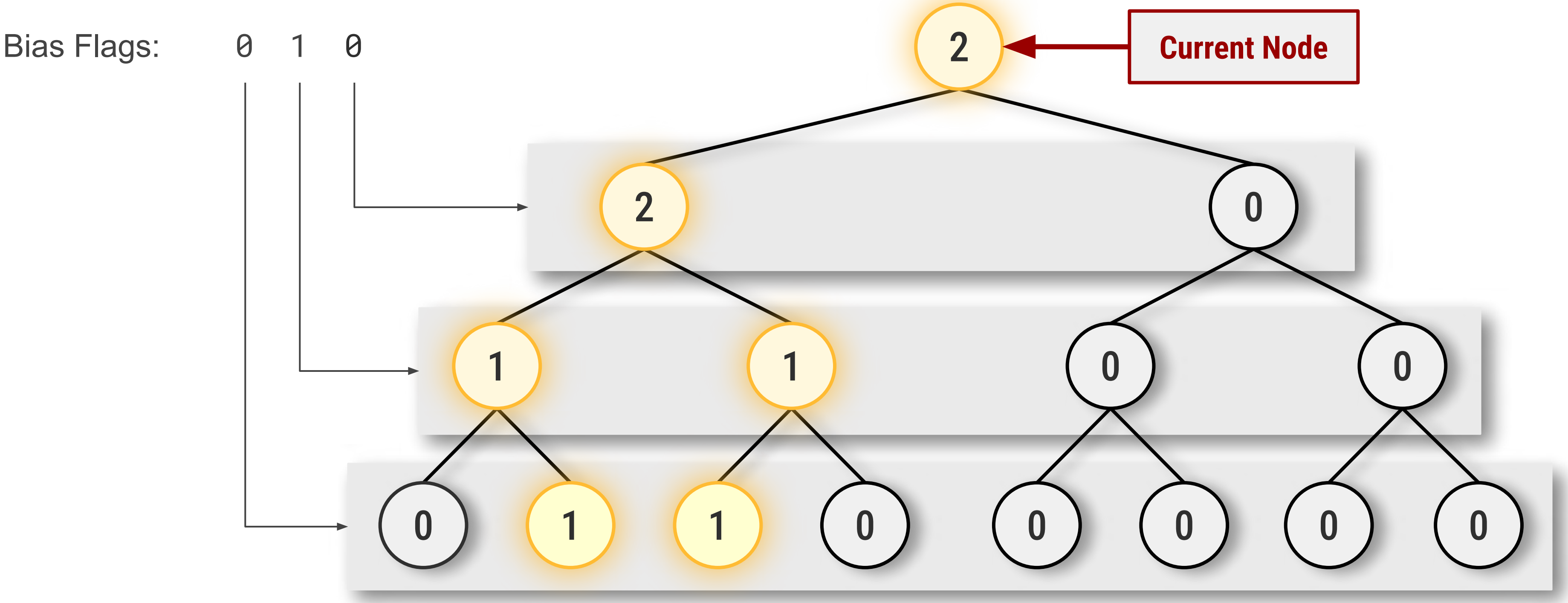
Bias: Fairness



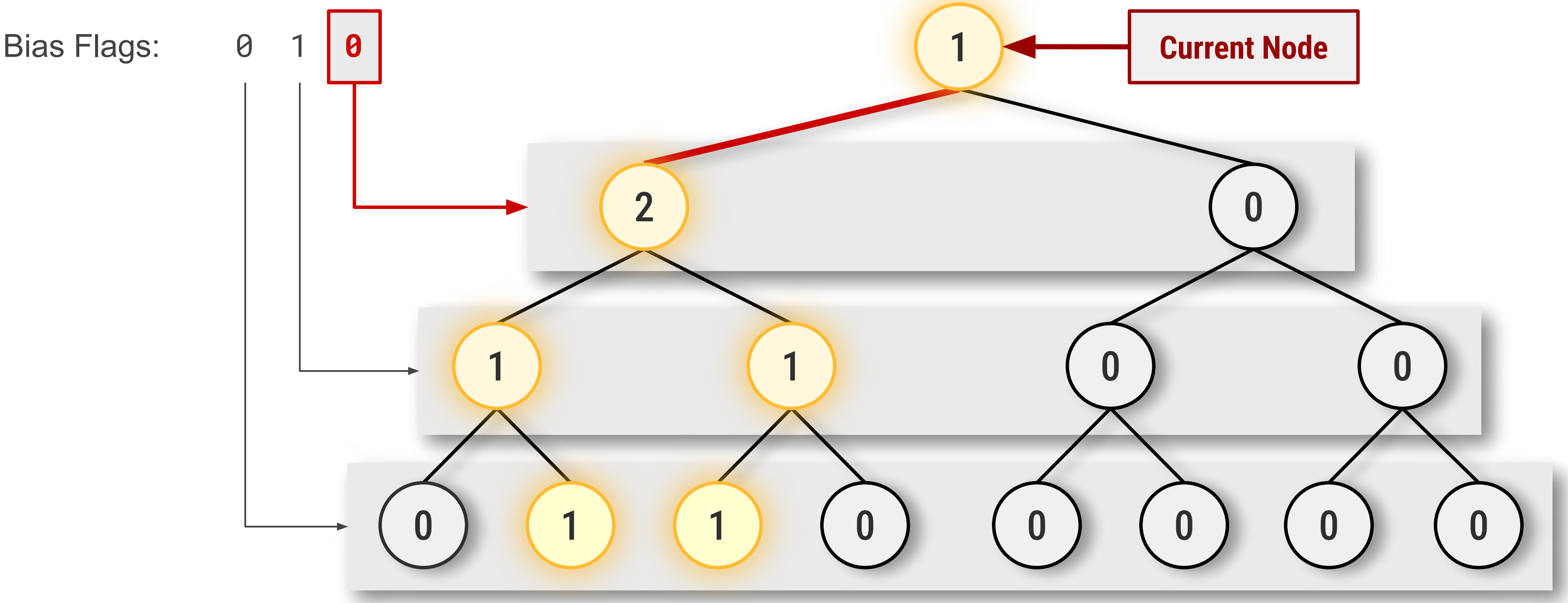
Bias: Fairness



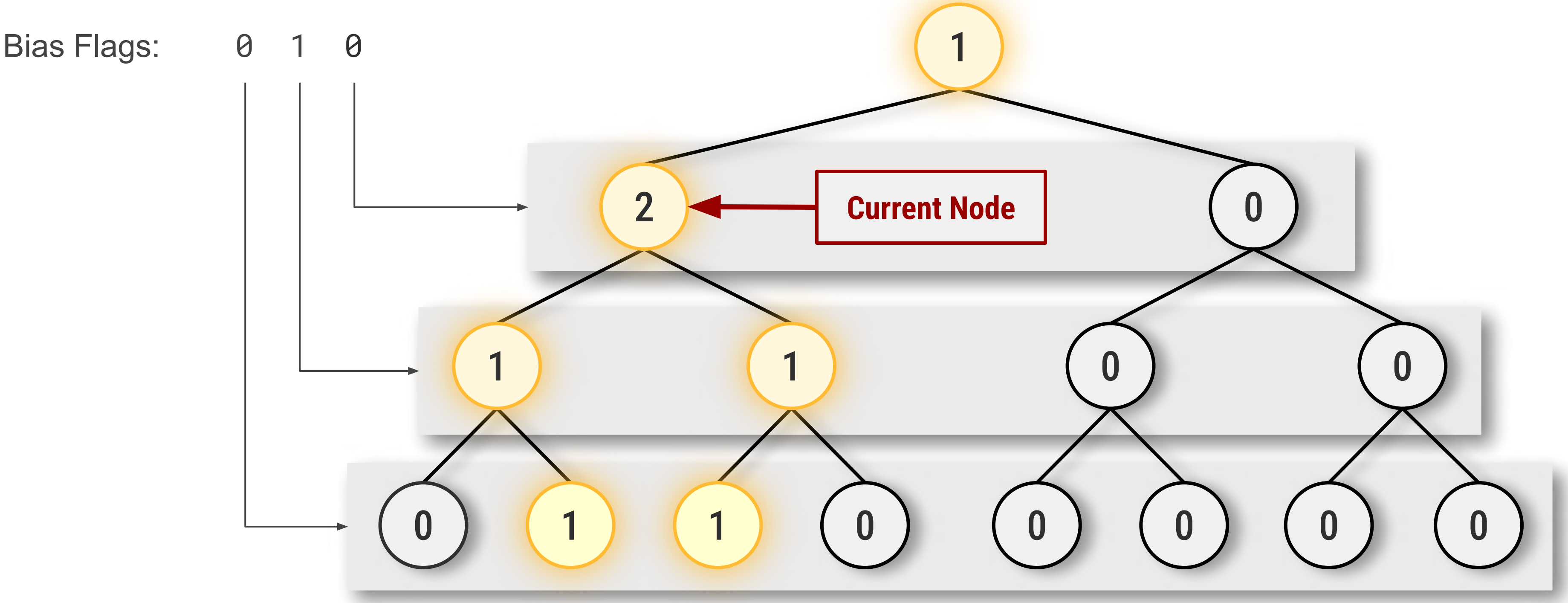
Bias: Fairness



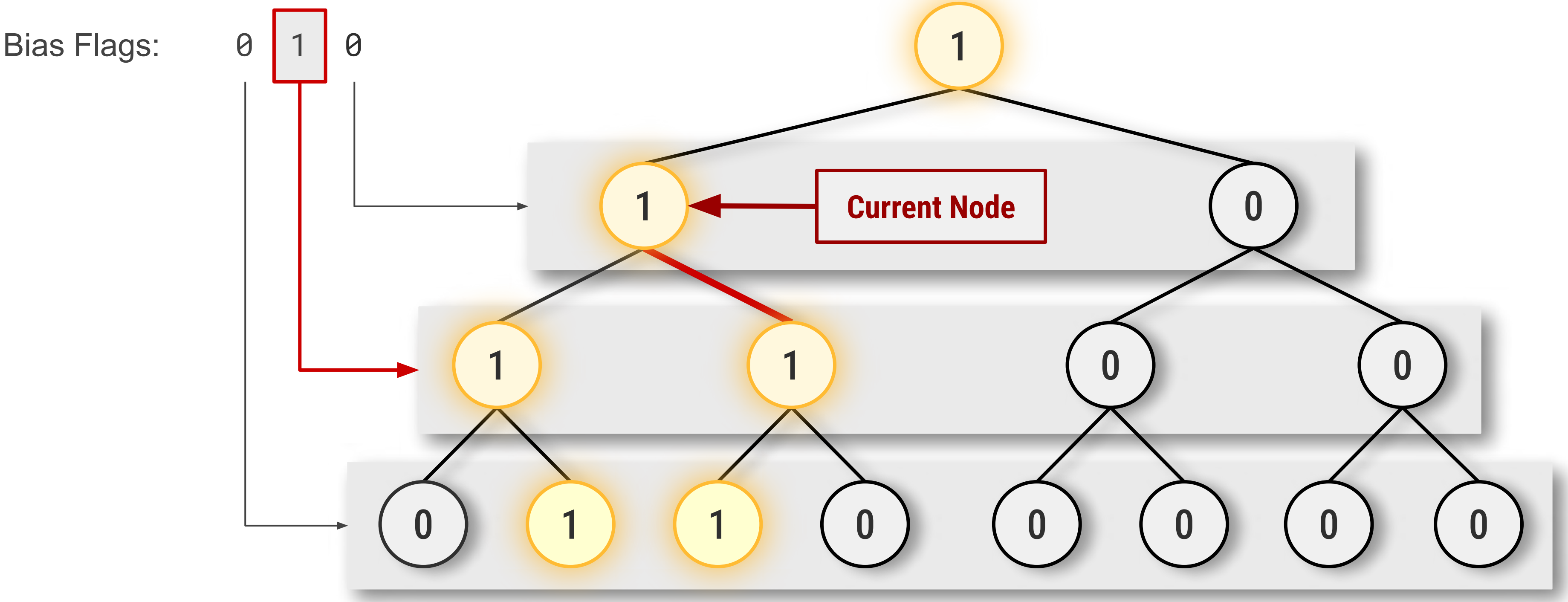
Bias: Fairness



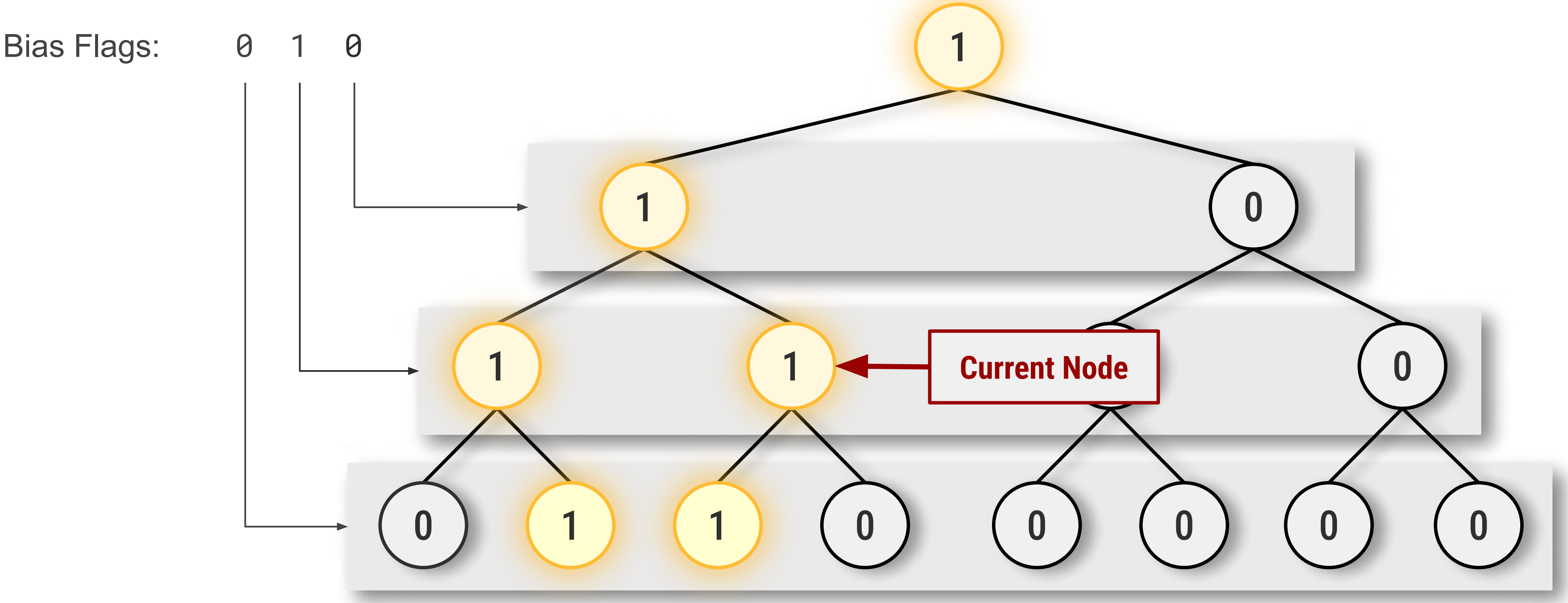
Bias: Fairness



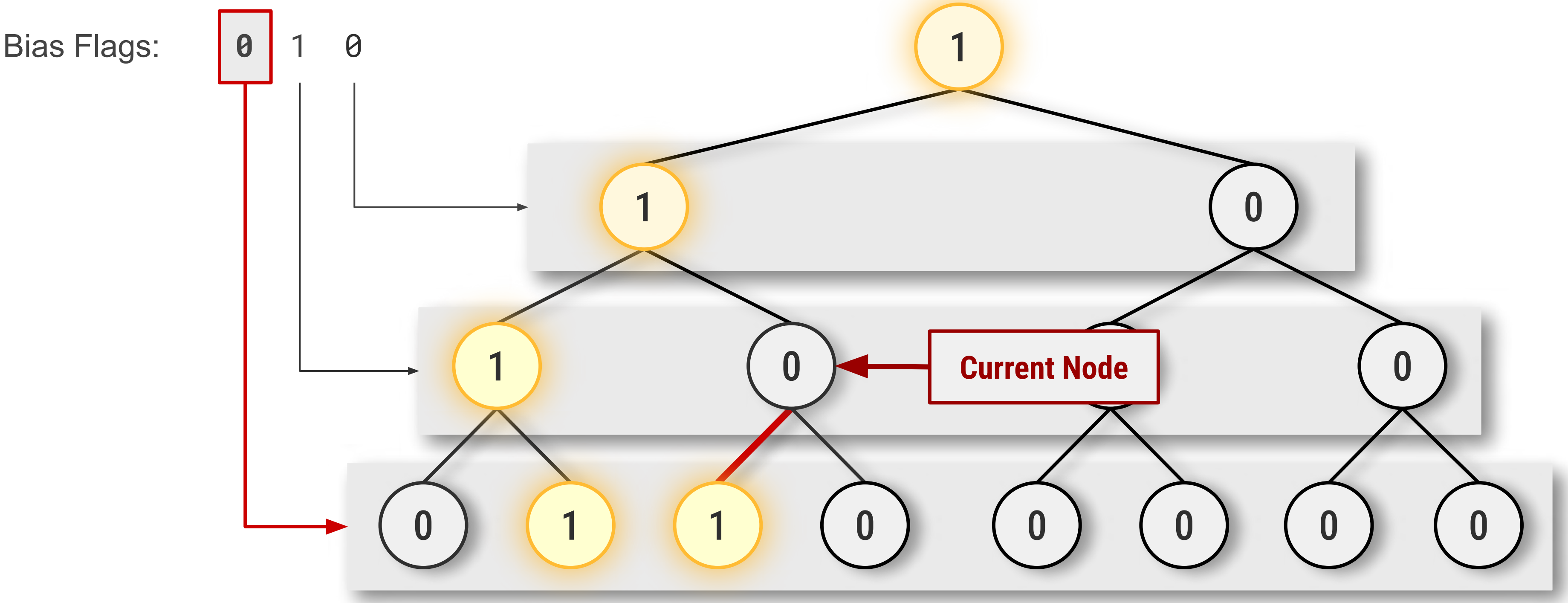
Bias: Fairness



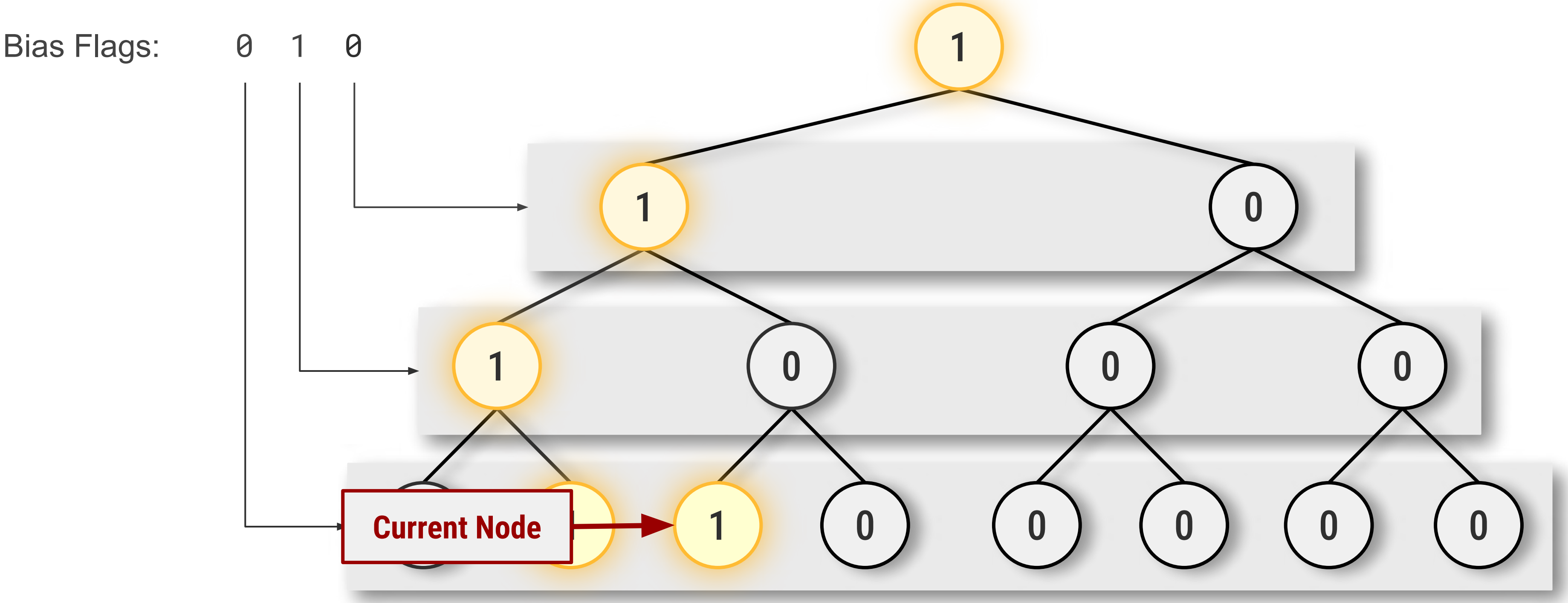
Bias: Fairness



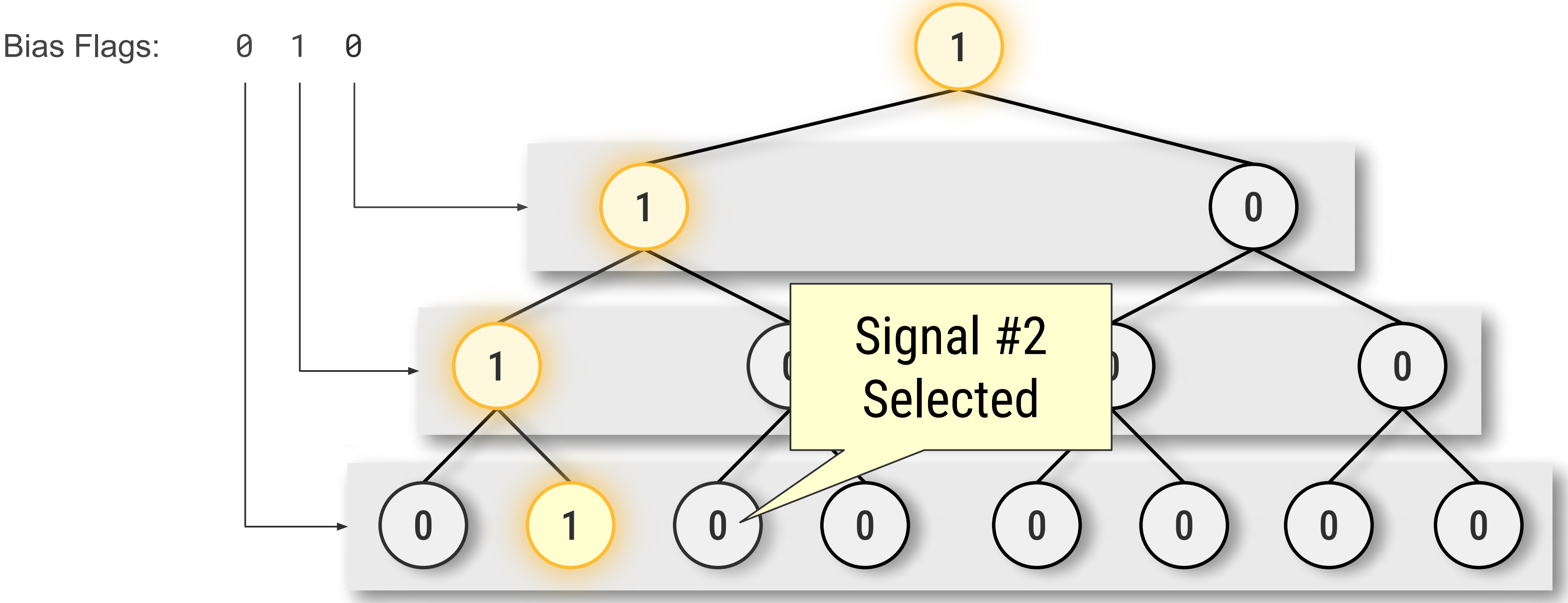
Bias: Fairness



Bias: Fairness



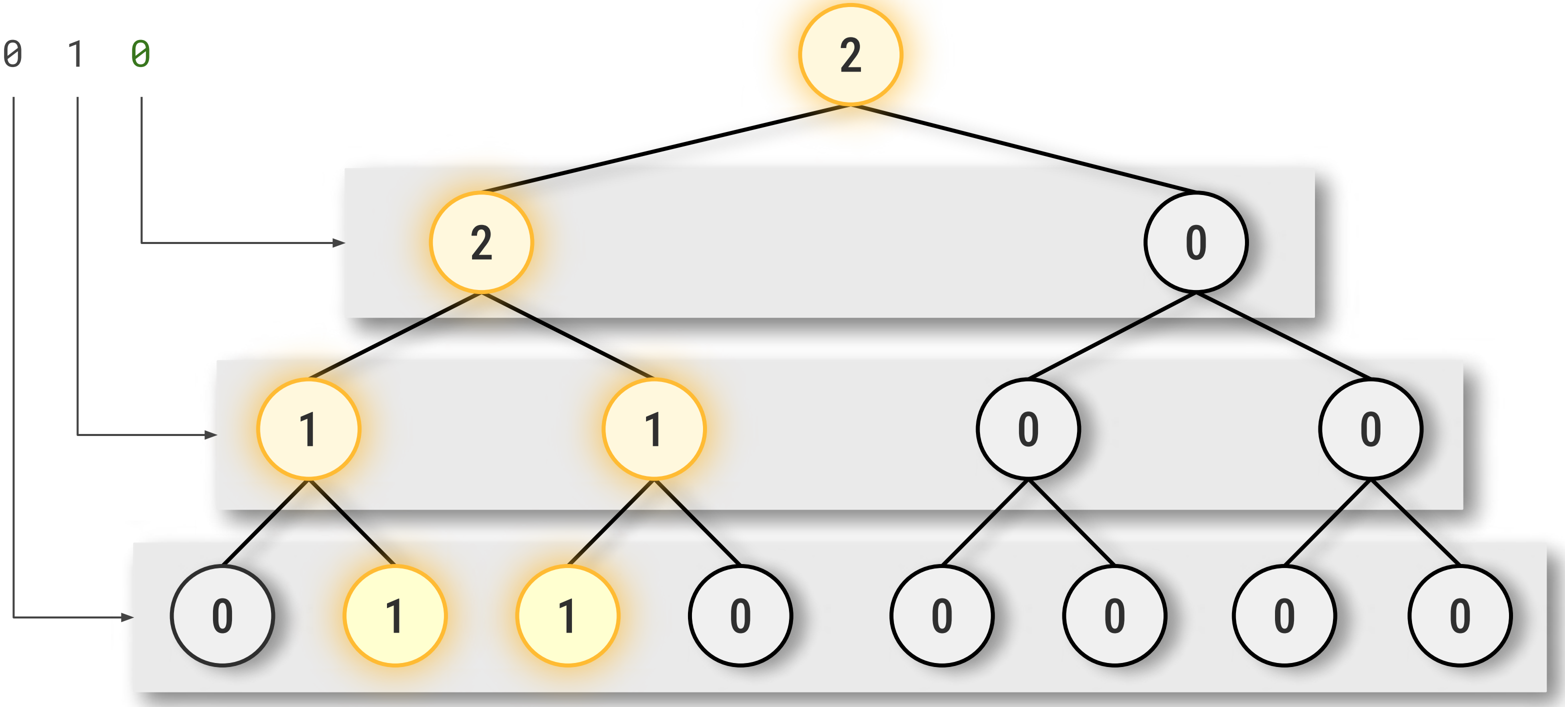
Bias: Fairness



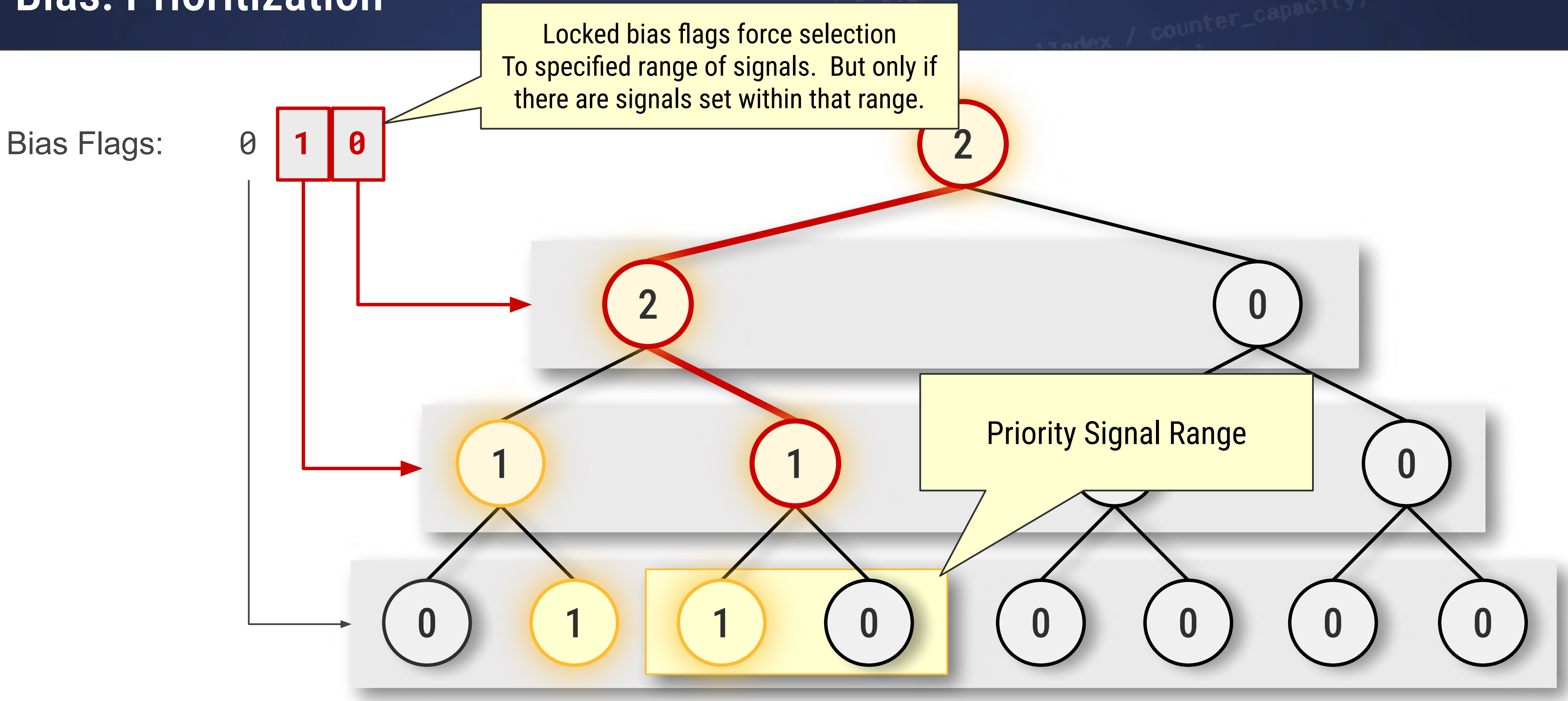
Signal #2 Selected

Bias: Prioritization

Bias Flags: `0 1 0`



Bias: Prioritization



Locked bias flags force selection
To specified range of signals. But only if
there are signals set within that range.

Priority Signal Range

Creating Bias:

Summary

- Excellent task fairness selection
- Zero overhead task prioritization
- Eliminates many issues related to queues entirely:
  - Avoids starvation
  - No need for custom solutions to allow prioritization
  - No need for task stealing
  - No need for multiple queues
- Allows “per thread” policy on selection strategy
  - Some worker threads might be ‘fair’
  - Others might be ‘priority’
  - Possible to dynamically adjust strategy
- Prioritization is opportunistic
  - Failure to locate priority signals seamlessly transitions back to fairness policy

Concurrency Based on Signal Trees

Work Contracts

Work Contracts:

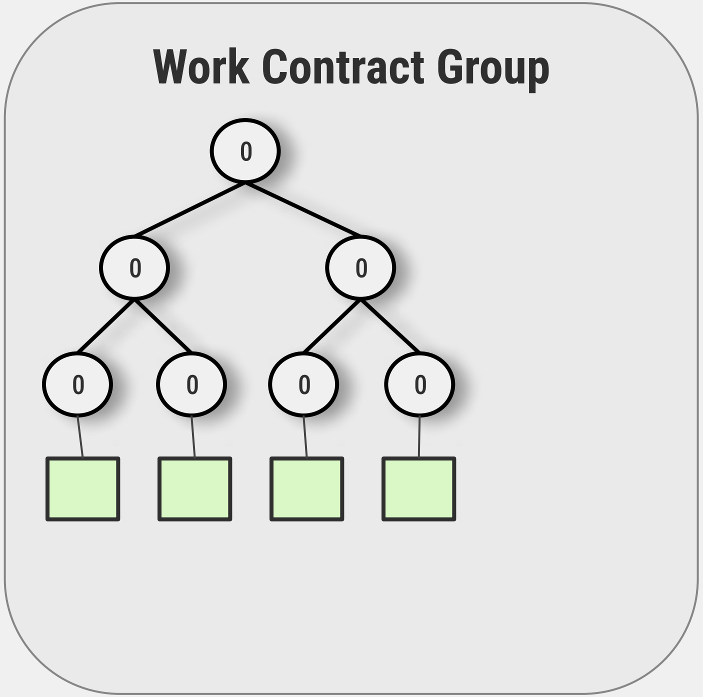
- Enhanced “Tasks” separating data from logic:
    - Contain its own logic
    - Asynchronous execution
    - Recurrening execution
    - Accesses its own data (user defined ingress)
    - Guaranteed single threaded execution
    - Supports optional asynchronous destruction logic

Basic Overview:

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“hello world!”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

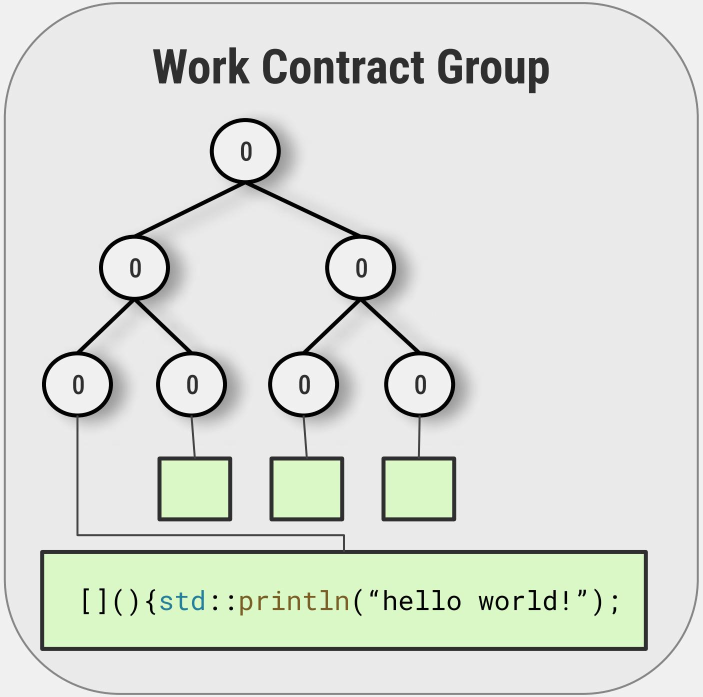
Basic Overview:

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println("hello world!");});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```



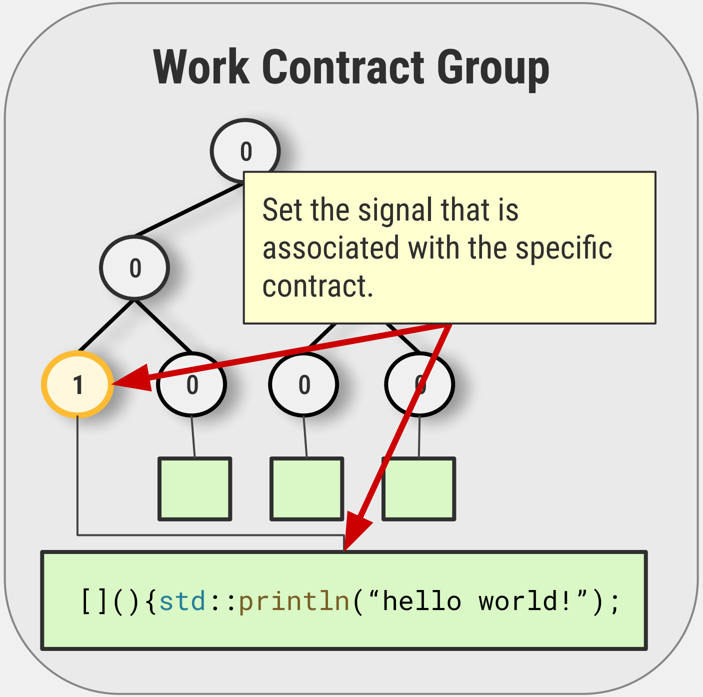
Basic Overview:

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“hello world!”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```



Basic Overview:

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“hello world!”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```



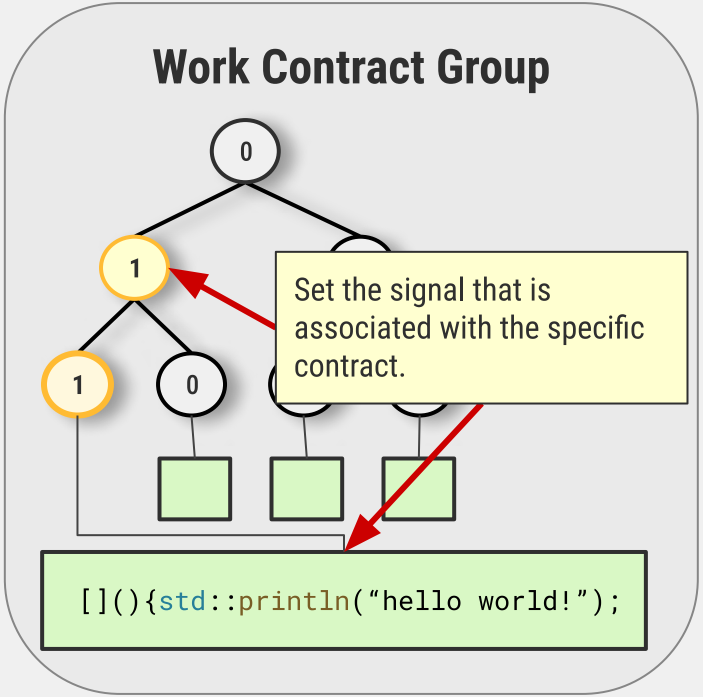
Work Contract Group

Set the signal that is associated with the specific contract.

```
[](){std::println(“hello world!”);
```

Basic Overview:

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“hello world!”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

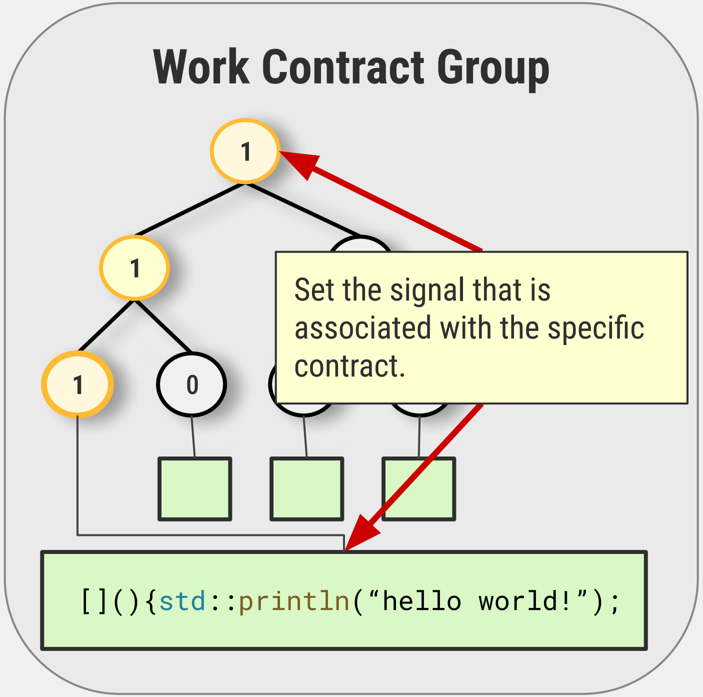


```
[](){std::println(“hello world!”);
```

Set the signal that is associated with the specific contract.

Basic Overview:

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“hello world!”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```



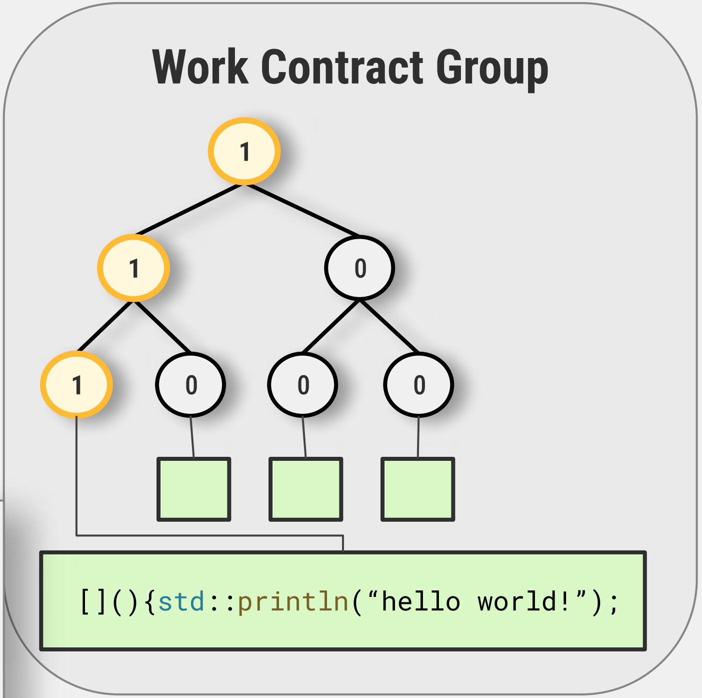
Set the signal that is associated with the specific contract.

```
[](){std::println(“hello world!”);
```

Basic Overview:

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println("hello world!");});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

```
void work_contract_group::execute_next_contract()
{
   auto signalIndex = signalTree_.select();
   contracts_[signalIndex].invoke();
}
```

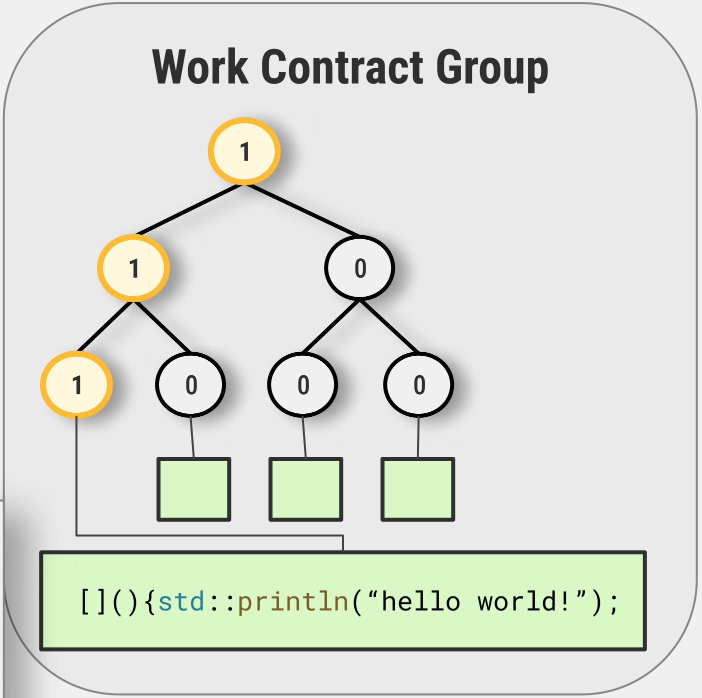


```
[](){std::println("hello world!");
```

Basic Overview:

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“hello world!”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

```
void work_contract_group::execute_next_contract()
{
   auto signalIndex = signalTree_.select();
   contracts_[signalIndex].invoke();
}
```

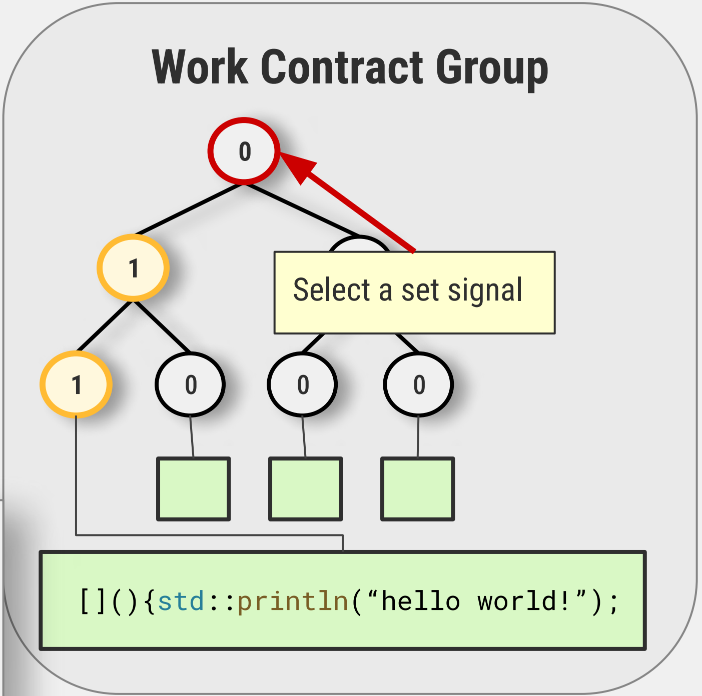


```
[](){std::println(“hello world!”);
```

Basic Overview:

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“hello world!”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

```
void work_contract_group::execute_next_contract()
{
   auto signalIndex = signalTree_.select();
   contracts_[signalIndex].invoke();
}
```



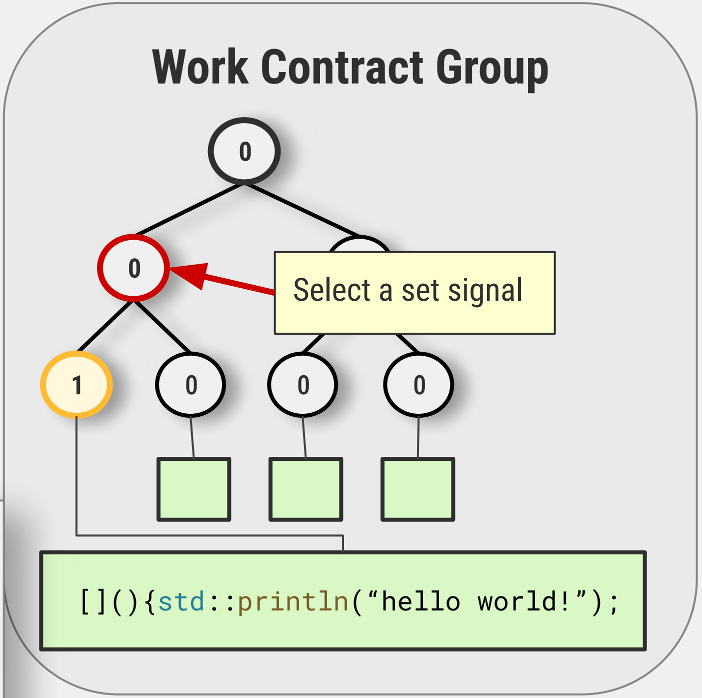
Work Contract Group

```
[](){std::println(“hello world!”);
```

Basic Overview:

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“hello world!”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

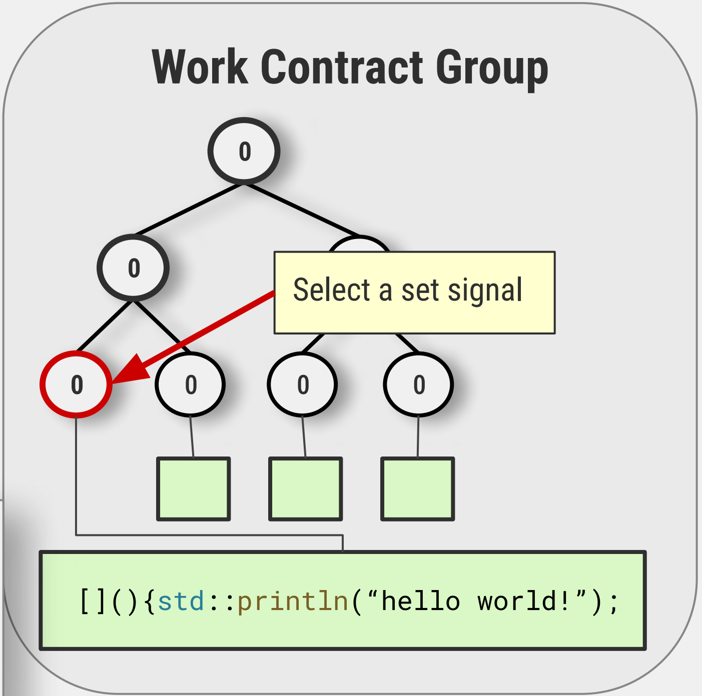
```
void work_contract_group::execute_next_contract()
{
   auto signalIndex = signalTree_.select();
   contracts_[signalIndex].invoke();
}
```



Basic Overview:

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println("hello world!");});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

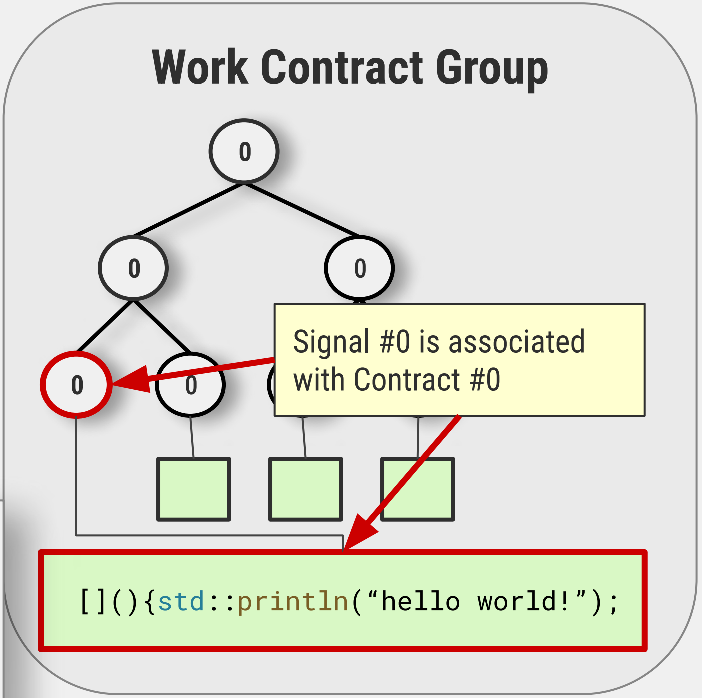
```
void work_contract_group::execute_next_contract()
{
   auto signalIndex = signalTree_.select();
   contracts_[signalIndex].invoke();
}
```



Basic Overview:

```cpp
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“hello world!”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

```cpp
void work_contract_group::execute_next_contract()
{
   auto signalIndex = signalTree_.select();
   contracts_[signalIndex].invoke();
}
```



Work Contract Group

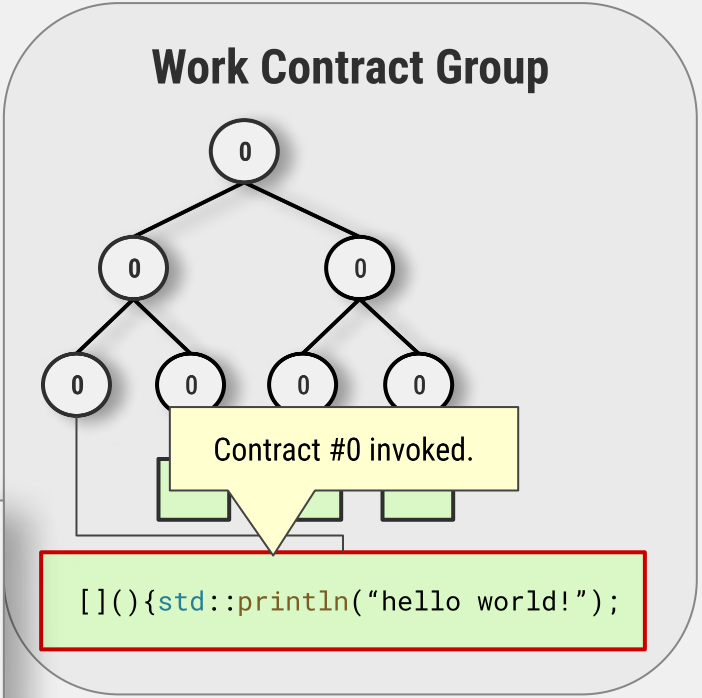
Signal #0 is associated with Contract #0

```cpp
[](){std::println(“hello world!”);
```

Basic Overview:

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println("hello world!");});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

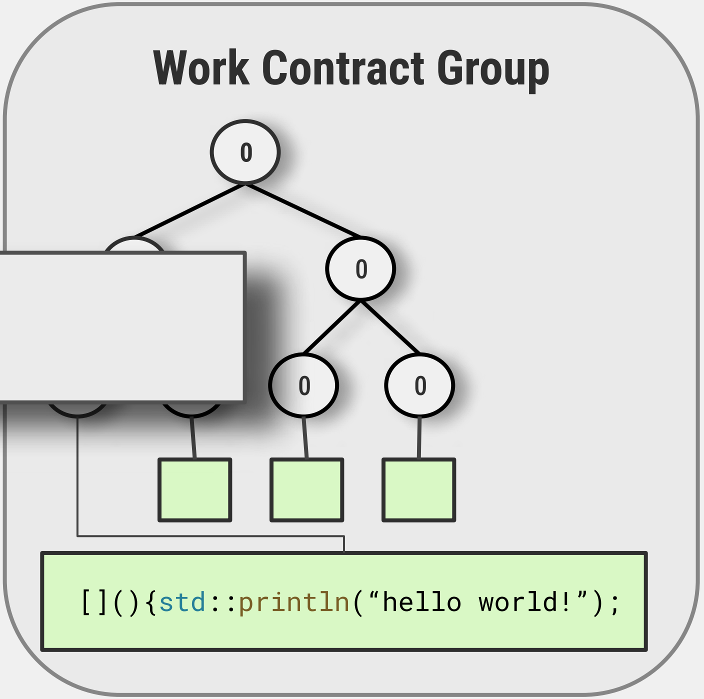
```
void work_contract_group::execute_next_contract()
{
   auto signalIndex = signalTree_.select();
   contracts_[signalIndex].invoke();
}
```



Basic Overview:

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“hello world!”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

```
hello world!
```



```
[](){std::println(“hello world!”);
```

Summary

- Work Contracts are managed by a parent Work Contract Group
  - The work contract group contains a signal tree
  - That signal tree has the same capacity as the as the number of contracts in the group
  - ‘Scheduling’ a work contract causes the associated signal to be set
  - Work Contracts are invoked via the parent Work Contract Group using work_contract_group::execute_next_contract();
  - When a work contract is invoked it is also ‘de-scheduled’ and its signal is reset

Work Contract

asynchronous ‘destructor’

Work Contract: Asynchronous Destruction

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“executed”);},
    [](){std::println(“destroyed”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();

// release the work contract
workContract.release();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

Work Contract: Asynchronous Destruction

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“executed”);},
    [](){std::println(“destroyed”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();

// release the work contract
workContract.release();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

Work Contract: Asynchronous Destruction

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“executed”);},
    [](){std::println(“destroyed”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();

// release the work contract
workContract.release();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

Work Contract: Asynchronous Destruction

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“executed”);},
    [](){std::println(“destroyed”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();

// release the work contract
workContract.release();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

Work Contract: Asynchronous Destruction

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“executed”);},
    [](){std::println(“destroyed”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();

// release the work contract
workContract.release();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

```
executed
```

Work Contract: Asynchronous Destruction

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“executed”);},
    [](){std::println(“destroyed”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();

// release the work contract
workContract.release();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

```
executed
```

Work Contract: Asynchronous Destruction

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [](){std::println(“executed”);},
    [](){std::println(“destroyed”);});

// schedule the work contract
workContract.schedule();

// execute the next scheduled contract
workContractGroup.execute_next_contract();

// release the work contract
workContract.release();

// execute the next scheduled contract
workContractGroup.execute_next_contract();
```

```
executed
destroyed
```

Summary

- Work Contracts support an optional “destructor”
- Releasing a work contract will schedule it for destruction
- Going out of scope also ‘releases’ the work contract
- The ‘destructor’ will be invoked asynchronously
- A ‘released’ work contract is automatically ‘scheduled’ as well

Work Contract

Repeatable Invocation

Work Contract: Repeatable Invocation

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [i = 0]() mutable
    {
        std::println(“i = {}”, i++);
    });

workContract.schedule();
workContractGroup.execute_next_contract();

workContract.schedule();
workContractGroup.execute_next_contract();
```

Work Contract: Repeatable Invocation

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [i = 0]() mutable
    {
        std::println(“i = {}”, i++);
    });

workContract.schedule();
workContractGroup.execute_next_contract();

workContract.schedule();
workContractGroup.execute_next_contract();
```

Work Contract: Repeatable Invocation

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [i = 0]() mutable
    {
        std::println(“i = {}”, i++);
    });

workContract.schedule();
workContractGroup.execute_next_contract();

workContract.schedule();
workContractGroup.execute_next_contract();
```

Work Contract: Repeatable Invocation

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [i = 0]() mutable
    {
        std::println(“i = {}”, i++);
    });

workContract.schedule();
workContractGroup.execute_next_contract();

workContract.schedule();
workContractGroup.execute_next_contract();
```

```
i = 1
```

Work Contract: Repeatable Invocation

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [i = 0]() mutable
    {
        std::println(“i = {}”, i++);
    });

workContract.schedule();
workContractGroup.execute_next_contract();

workContract.schedule();
workContractGroup.execute_next_contract();
```

```
i = 1
```

Work Contract: Repeatable Invocation

```
// create a work contract group
work_contract_group workContractGroup(4);

// create work work contract
auto workContract =
    workContractGroup.create_contract(
    [i = 0]() mutable
    {
        std::println(“i = {}”, i++);
    });

workContract.schedule();
workContractGroup.execute_next_contract();

workContract.schedule();
workContractGroup.execute_next_contract();
```

```
i = 1
i = 2
```

Summary

- Work Contracts are intended to be long lived objects
- They can be repeatedly re-scheduled and re-invoked

Work Contract

work_contract_token

Work Contract: Work Contract Token

```
// create a work contract group
work_contract_group workContractGroup(4);
std::atomic<bool> done{false};

// create work work contract
auto workContract = workContractGroup.create_contract(
      [i = 0](work_contract_token & token) mutable
      {
          std::println(“i = {}”, i++);
          if (i < 2)
              token.schedule();
          else
              token.release();
      },
      [&](){std::println(“destroyed”); done = true;});

workContract.schedule();
while (!done)
    workContractGroup.execute_next_contract();
```

Work Contract: Work Contract Token

```
// create a work contract group
work_contract_group workContractGroup(4);
std::atomic<bool> done{false};

// create work work contract
auto workContract = workContractGroup.create_contract(
      [i = 0](work_contract_token & token) mutable
      {
          std::println(“i = {}”, i++);
          if (i < 2)
              token.schedule();
          else
              token.release();
      },
      [&](){std::println(“destroyed”); done = true;});

workContract.schedule();
while (!done)
    workContractGroup.execute_next_contract();
```

Work Contract: Work Contract Token

```
// create a work contract group
work_contract_group workContractGroup(4);
std::atomic<bool> done{false};

// create work work contract
auto workContract = workContractGroup.create_contract(
      [i = 0](work_contract_token & token) mutable
      {
          std::println(“i = {}”, i++);
          if (i < 2)
              token.schedule();
          else
              token.release();
      },
      [&](){std::println(“destroyed”); done = true;});

workContract.schedule();
while (!done)
    workContractGroup.execute_next_contract();
```

Work Contract: Work Contract Token

```
// create a work contract group
work_contract_group workContractGroup(4);
std::atomic<bool> done{false};

// create work work contract
auto workContract = workContractGroup.create_contract(
      [i = 0](work_contract_token & token) mutable
      {
          std::println(“i = {}”, i++);
          if (i < 2)
              token.schedule();
          else
              token.release();
      },
      [&](){std::println(“destroyed”); done = true;});

workContract.schedule();
while (!done)
    workContractGroup.execute_next_contract();
```

```
i = 0
```

Work Contract: Work Contract Token

```
// create a work contract group
work_contract_group workContractGroup(4);
std::atomic<bool> done{false};

// create work work contract
auto workContract = workContractGroup.create_contract(
      [i = 0](work_contract_token & token) mutable
      {
          std::println("i = {}", i++);
          if (i < 2)
              token.schedule();
          else
              token.release();
      },
      [&](){std::println("destroyed"); done = true;});

workContract.schedule();
while (!done)
    workContractGroup.execute_next_contract();
```

```
i = 0
i = 1
```

Work Contract: Work Contract Token

```
// create a work contract group
work_contract_group workContractGroup(4);
std::atomic<bool> done{false};

// create work work contract
auto workContract = workContractGroup.create_contract(
      [i = 0](work_contract_token & token) mutable
      {
          std::println("i = {}", i++);
          if (i < 2)
              token.schedule();
          else
              token.release();
      },
      [&](){std::println("destroyed"); done = true;});

workContract.schedule();
while (!done)
    workContractGroup.execute_next_contract();
```

```
i = 0
i = 1
destroyed
```

Work Contract

Adding Threading

Work Contract: Adding Threading

```
work_contract_group workContractGroup;

std::vector<std::jthread> threads(24);
for (auto & thread : threads)
    thread = std::move(std::jthread([&](auto stopToken)
    {while (!stopToken.stop_requested()) workContractGroup.execute_next_contract();}));

// create work work contract
auto workContract = workContractGroup.create_contract(
    [i = 0](auto & token) mutable{std::cout << "thread " << std::this_thread::get_id() <<
    ", i = " << i++ << '\n'; token.schedule();});

workContract.schedule();
std::this_thread::sleep_for(std::chrono::seconds(10));
```

Work Contract: Adding Threading

```
work_contract_group workContractGroup;

std::vector<std::jthread> threads(24);
for (auto & thread : threads)
    thread = std::move(std::jthread([&](auto stopToken)
    {while (!stopToken.stop_requested()) workContractGroup.execute_next_contract();}));

// create work work contract
auto workContract = workContractGroup.create_contract(
    [i = 0](auto & token) mutable{std::cout << "thread " << std::this_thread::get_id() <<
    ", i = " << i++ << '\n'; token.schedule();});

workContract.schedule();
std::this_thread::sleep_for(std::chrono::seconds(10));
```

Work Contract: Adding Threading

```
work_contract_group workContractGroup;

std::vector<std::jthread> threads(24);
for (auto & thread : threads)
    thread = std::move(std::jthread([&](auto stopToken)
    {while (!stopToken.stop_requested()) workContractGroup.execute_next_contract();}));

// create work work contract
auto workContract = workContractGroup.create_contract(
    [i = 0](auto & token) mutable{std::cout << "thread " << std::this_thread::get_id() <<
    ", i = " << i++ << '\n'; token.schedule();});

workContract.schedule();
std::this_thread::sleep_for(std::chrono::seconds(10));
```

Work Contract: Adding Threading

```
work_contract_group workContractGroup;

std::vector<std::jthread> threads(24);
for (auto & thr
    thread = st                                                      ract();}));

// create wo
auto workContra
    [i = 0](aut                                                      ::get_id() <<
    ", i = " <<

workContract.sc
std::this_threa
```

```
thread 130456429266624, i = 0
thread 130456450238144, i = 1
   ead 130456618010304, i = 2
   ead 130456618010304, i = 3
  read 130456450238144, i = 5
thread 130456397809344, i = 6
thread 130456576067264, i = 7
thread 130456597038784, i = 8
thread 130456429266624, i = 9
thread 130456471209664, i = 10
thread 130456576067264, i = 11
thread 130456597038784, i = 12
thread 130456429266624, i = 13
```

Different threads

No mangling
of the output

Work Contract: Single Threaded Execution

```cpp
spsc_fixed_queue<int> queue(1024);
work_contract_group workContractGroup;

static auto constexpr num_worker_threads = 16;
std::vector<std::jthread> workerThreads(num_worker_threads);
for (auto & workerThread : workerThreads)
    workerThread = std::jthread([&](auto token){while (!token.stop_requested())
    workContractGroup.execute_next_contract();});

// async consume
auto workContract = workContractGroup.create_contract([&](auto & token)
    {std::cout << "thread " << std::this_thread::get_id() << " consumed " << queue.pop() << "\n";
     if (!queue.empty()) token.schedule();});

// produce
for (auto i = 0; i < 8192; ++i)
    {while (!queue.push(i)); workContract.schedule();}

while (!queue.empty())
    ;
```

Work Contract: Single Threaded Execution

```
spsc_fixed_queue<int> queue(1024);
work_contract_group workContractGroup;

static auto con
std::vector<std
                                                                     )
    workCont

// async consum
auto workContra
    {std::cout                                                           queue.pop() << "\n";
     if (!queue
// produce

for (auto i = 0                                                      e();}
while (!queue.e
    ;
return 0;
```

```
thread 131483622704832 consumed 0
thread 131483654162112 consumed 1
thread 131483685619392 consumed 2
thread 131483580761792 consumed 3
thread 131483612219072 consumed 4
thread 131483591247552 consumed 5
thread 131483675133632 consumed 6
thread 131483675133632 consumed 7
thread 131483696105152 consumed 8
thread 131483675133632 consumed 9
thread 131483685619392 consumed 10
thread 131483664647872 consumed 11
thread 131483601733312 consumed 12
```

Different threads

No mangling
of the output

Talk Summary:

Summary

- Summarize that three principal techniques were introduced. Signal trees, selection bias and work contracts.
- Signal trees - light weight, scalable (easily support millions of signals), lock free/wait free, replacement for task queues, incredibly fast, vastly outperforms MPMC queues
- Bias - excellent task fairness selection, eliminates need for priority queues, eliminates potential for task starvation, etc. Can also be used to create task prioritization with same guarantees re: task starvation
- Work Contracts - build on Signal Trees, easy to use, simply design, powerful toolkit for building concurrency and parallelism.

Thank You

For listening

Thank You

And thank you to Neva, Dar, Anni, Andrei and Lime Trading

LIME
TRADING

Questions?

Work Contracts

Rethinking Task Based Concurrency and Parallelism for Low Latency C++

MICHAEL A MANISCALCO

Cppcon The C++ Conference

2024 September 15 - 20

Resources:

Source Code:

github.com/buildingcpp/work_contract

This Talk:

cppcon2024.sched.com

Contact:

wc@michael-maniscalco.com

Lime Trading:

Lime.co

LIME TRADING