Cloud9

Parallel Symbolic Execution for Automated Real-World Software Testing

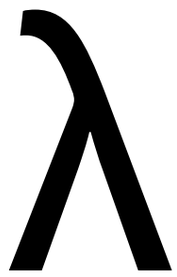
Stefan Bucur, Vlad Ureche, Cristian Zamfir, George Candea

School of Computer and Communication Sciences

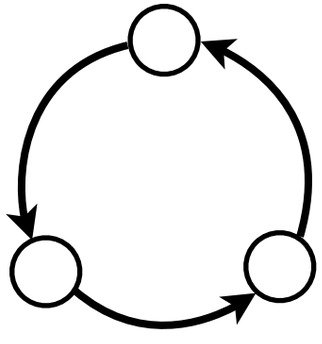
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Automated Software Testing

Automated Techniques



Symbolic Execution



Model Checking

Scalability

Applicability

Usability

Industrial SW Testing

Manual Testing

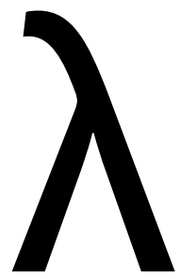
Static Analysis

Fuzzing

Cloud9 - The Big Picture

- Parallel symbolic execution
  - *Linear scalability on commodity clusters*
- Full symbolic POSIX support
  - *Applicable on real-world systems*
- Platform for writing test cases
  - *Easy-to-use platform API*

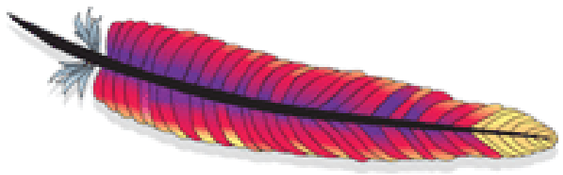
Automated Systems Testing



Symbolic Execution

- Promising for systems testing: KLEE [*]
- High-coverage test cases
- Found new bugs
- ... But applied only on small programs

[*] C. Cadar, D. Dunbar, D. Engler, *"KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs"*, OSDI 2008



Apache

Memcached

GNU Coreutils

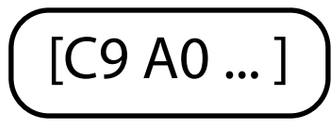
Symbolic Execution in a Nutshell

```
void proc_pkt(packet_t* pkt) {
  if (pkt->magic != 0xC9) {
    err(pkt);
    return;
  }
  if (pkt->cmd == GET) {
    ...
  } else if ...
  ...
}
```

[C9 A0 ...]

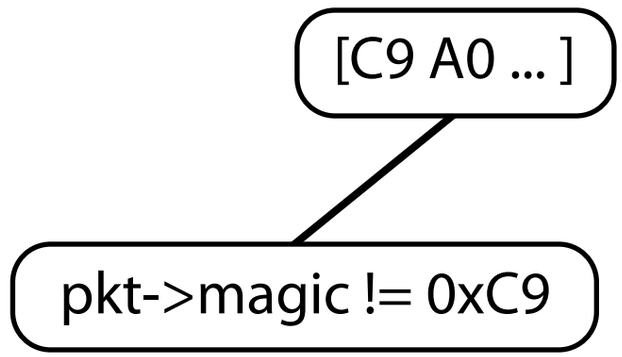
Symbolic Execution in a Nutshell

```
void proc_pkt(packet_t* pkt) {
  if (pkt->magic != 0xC9) {
    err(pkt);
    return;
  }
  if (pkt->cmd == GET) {
    ...
  } else if ...
  ...
}
```



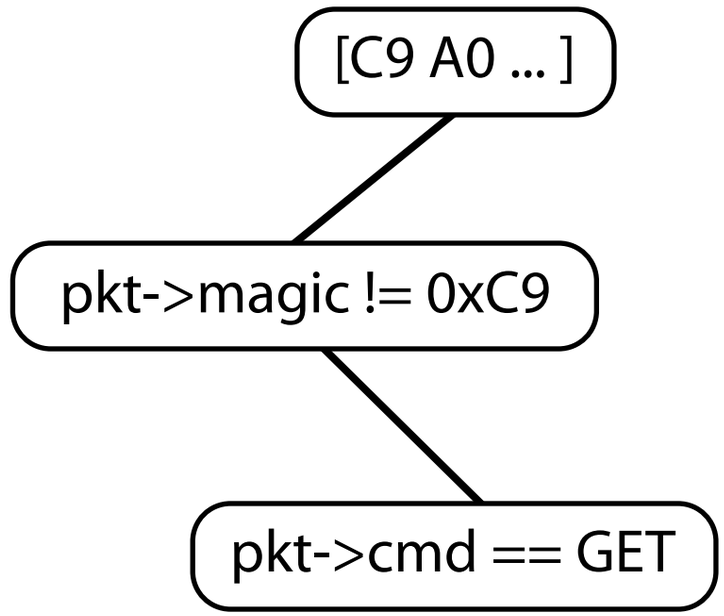
Symbolic Execution in a Nutshell

```
void proc_pkt(packet_t* pkt) {
  if (pkt->magic != 0xC9) {
    err(pkt);
    return;
  }
  if (pkt->cmd == GET) {
    ...
  } else if ...
  ...
}
```



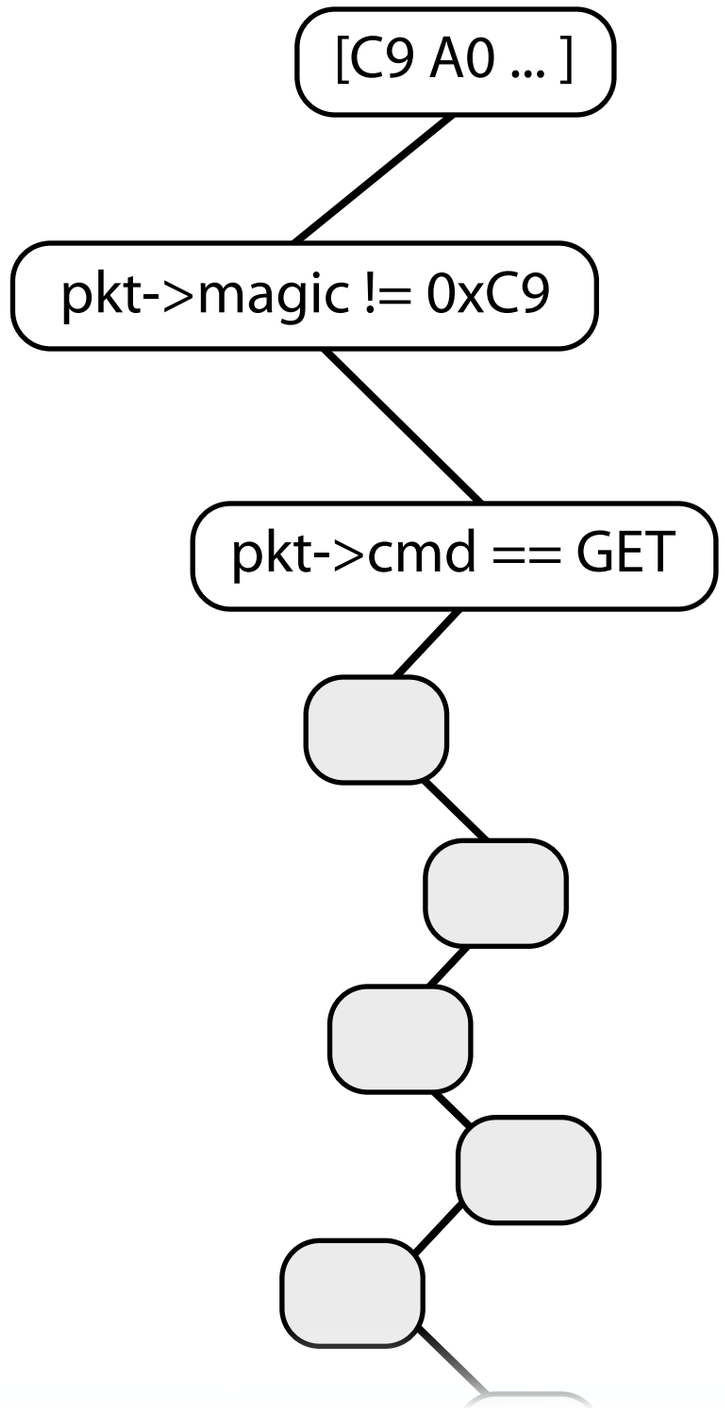
Symbolic Execution in a Nutshell

```
void proc_pkt(packet_t* pkt) {
  if (pkt->magic != 0xC9) {
    err(pkt);
    return;
  }
  if (pkt->cmd == GET) {
    ...
  } else if ...
  ...
}
```



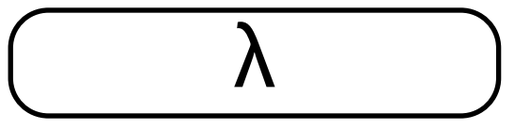
Symbolic Execution in a Nutshell

```
void proc_pkt(packet_t* pkt) {
  if (pkt->magic != 0xC9) {
    err(pkt);
    return;
  }
  if (pkt->cmd == GET) {
    ...
  } else if ...
  ...
}
```



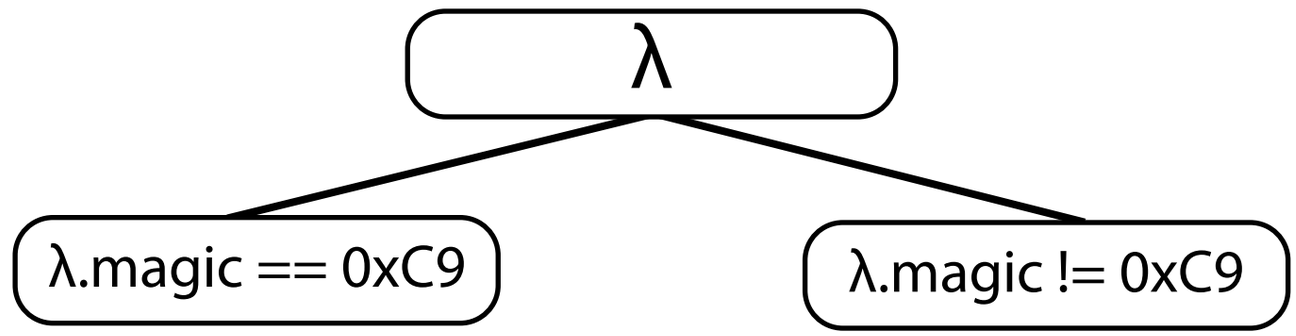
Symbolic Execution in a Nutshell

```
void proc_pkt(packet_t* pkt) {
  if (pkt->magic != 0xC9) {
    err(pkt);
    return;
  }
  if (pkt->cmd == GET) {
    ...
  } else if ...
  ...
}
```



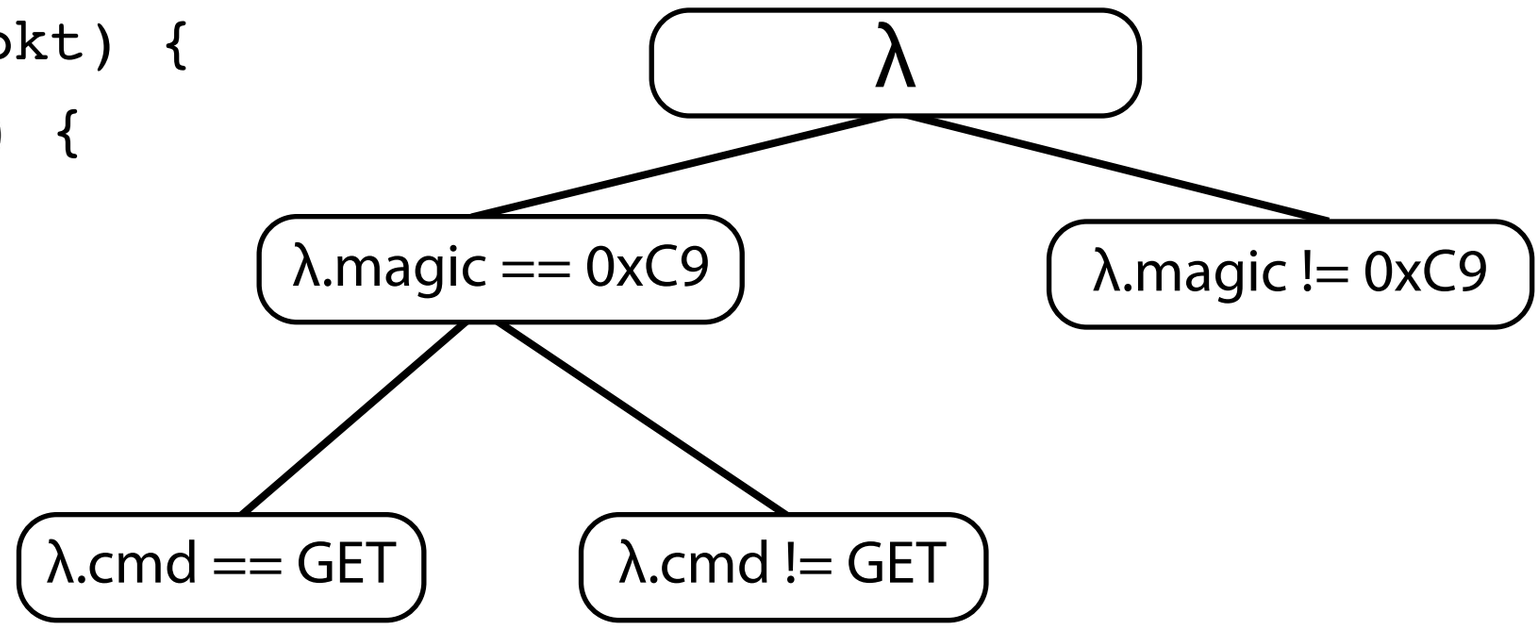
Symbolic Execution in a Nutshell

```
void proc_pkt(packet_t* pkt) {
  if (pkt->magic != 0xC9) {
    err(pkt);
    return;
  }
  if (pkt->cmd == GET) {
    ...
  } else if ...
  ...
}
```



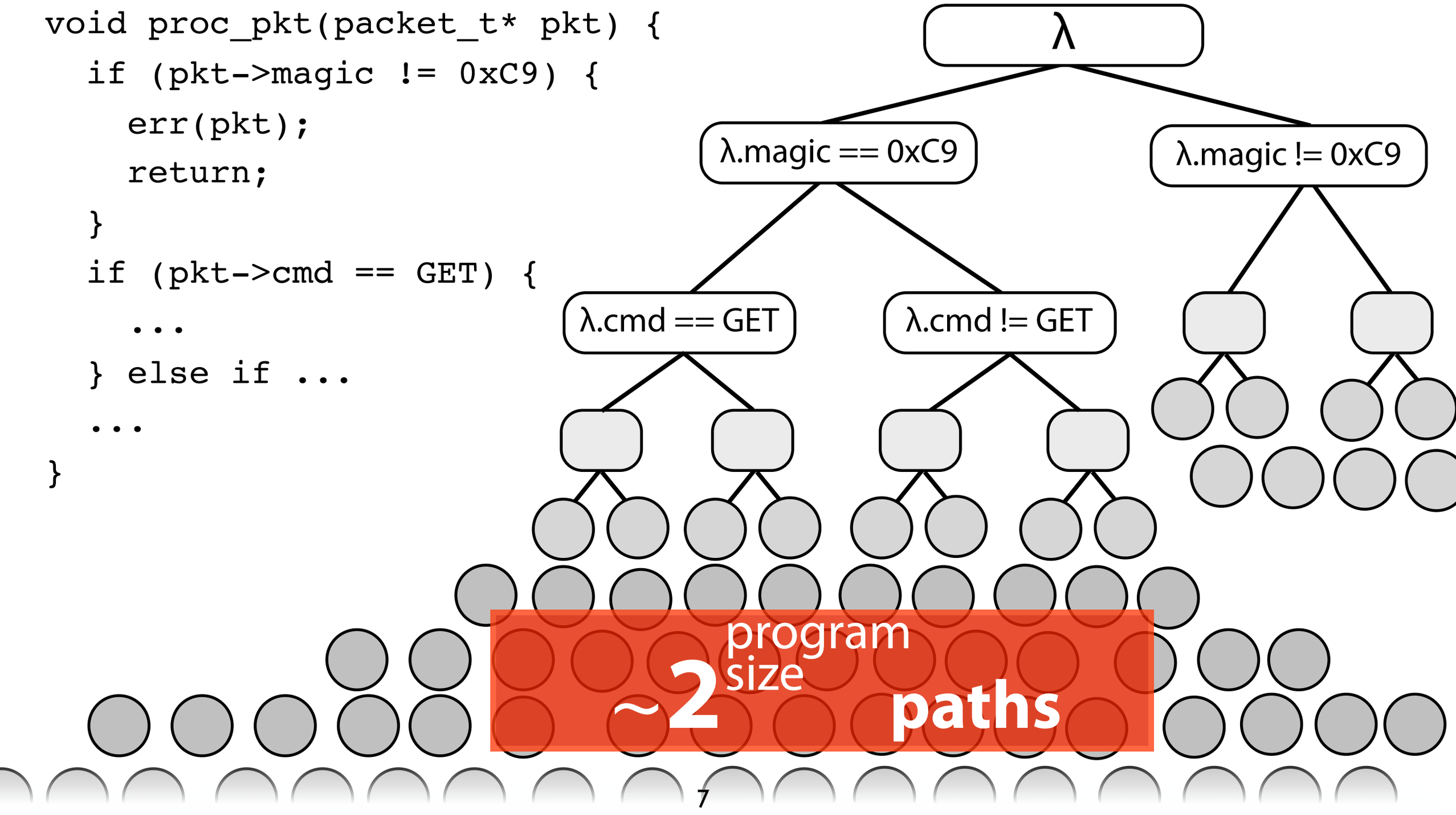
Symbolic Execution in a Nutshell

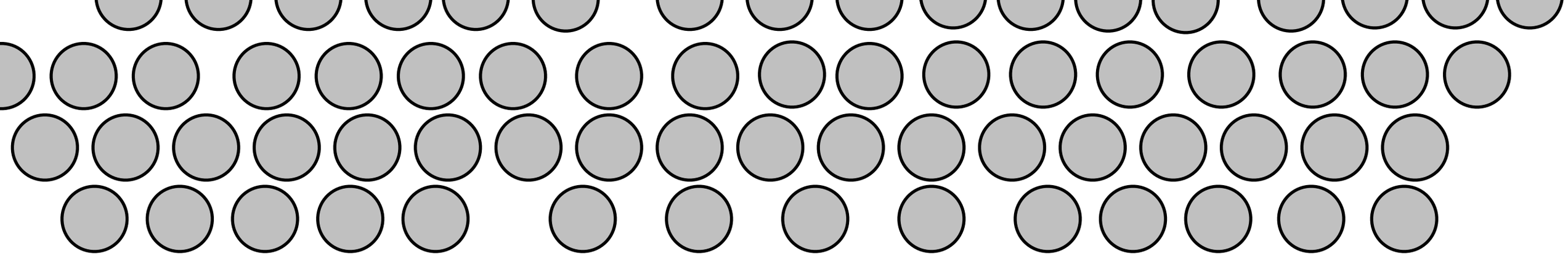
```
void proc_pkt(packet_t* pkt) {
  if (pkt->magic != 0xC9) {
    err(pkt);
    return;
  }
  if (pkt->cmd == GET) {
    ...
  } else if ...
  ...
}
```



Symbolic Execution in a Nutshell

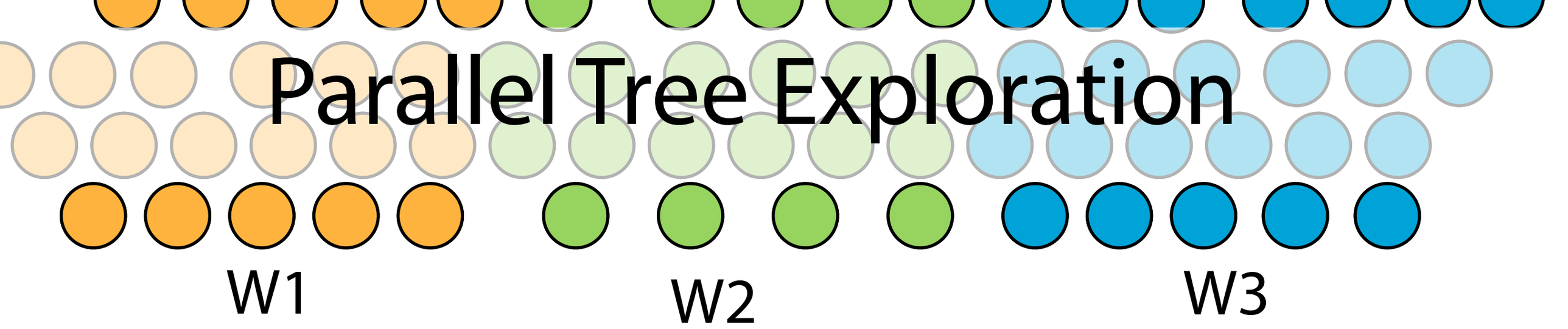
```
void proc_pkt(packet_t* pkt) {
  if (pkt->magic != 0xC9) {
    err(pkt);
    return;
  }
  if (pkt->cmd == GET) {
    ...
  } else if ...
  ...
}
```



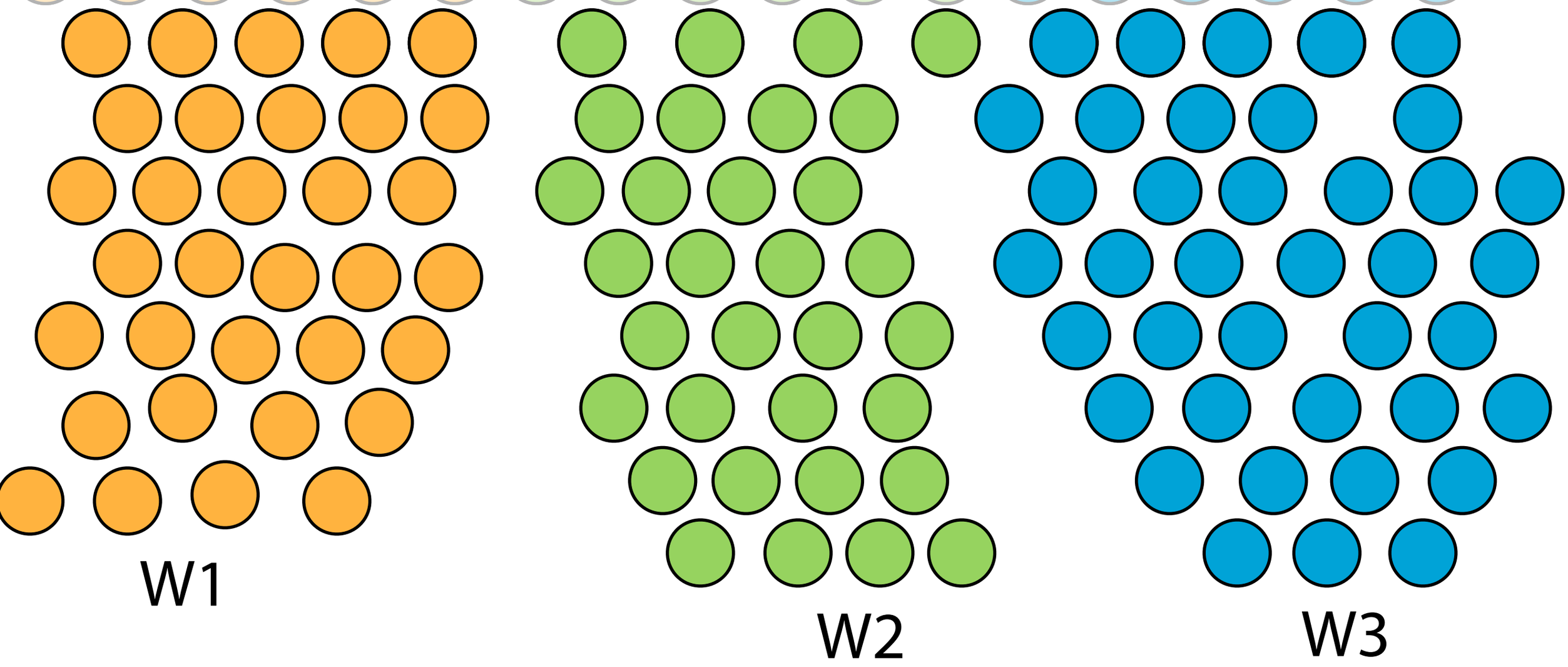


CPU Bottleneck
Memory Exhaustion

Parallel Tree Exploration

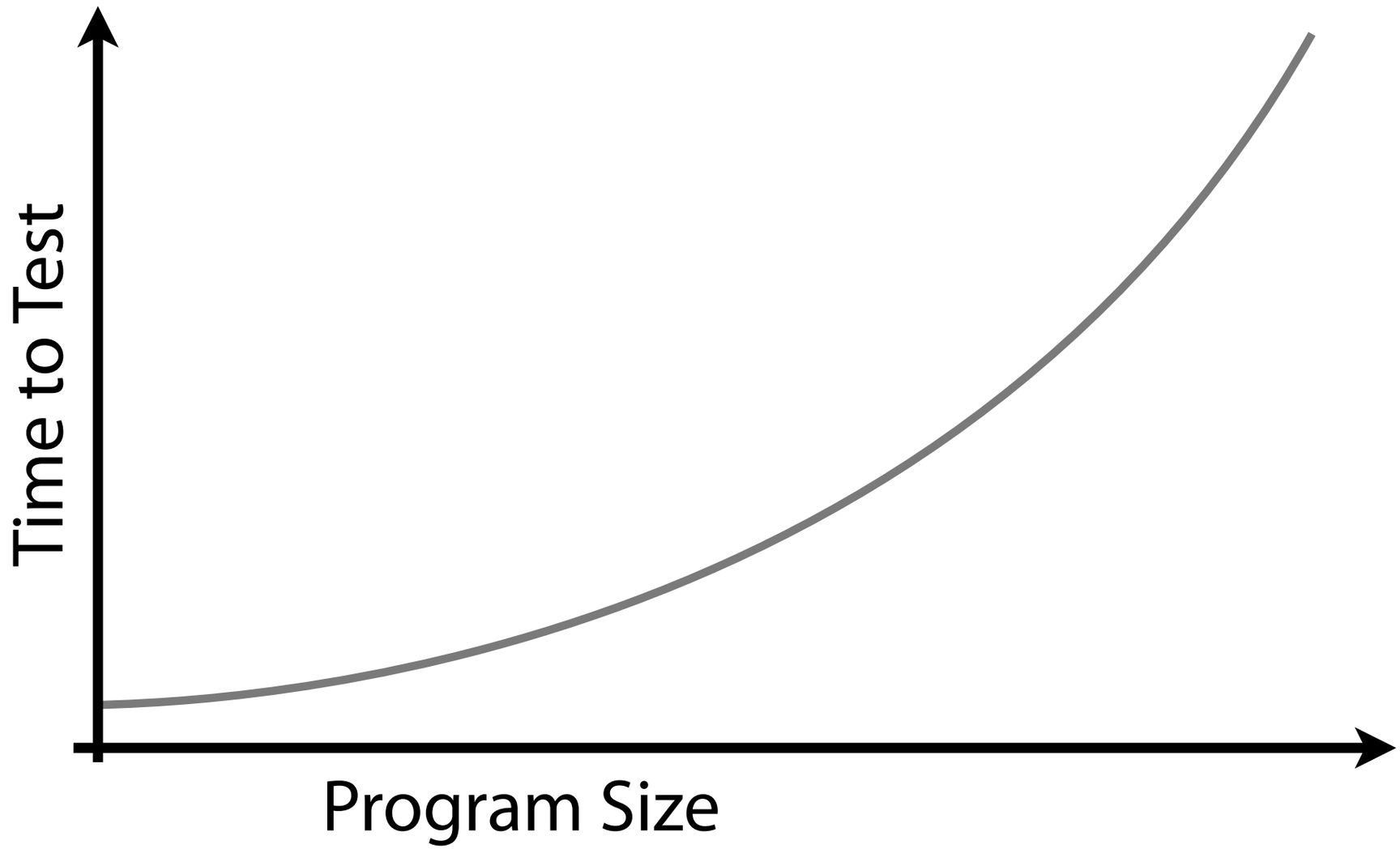


Parallel Tree Exploration

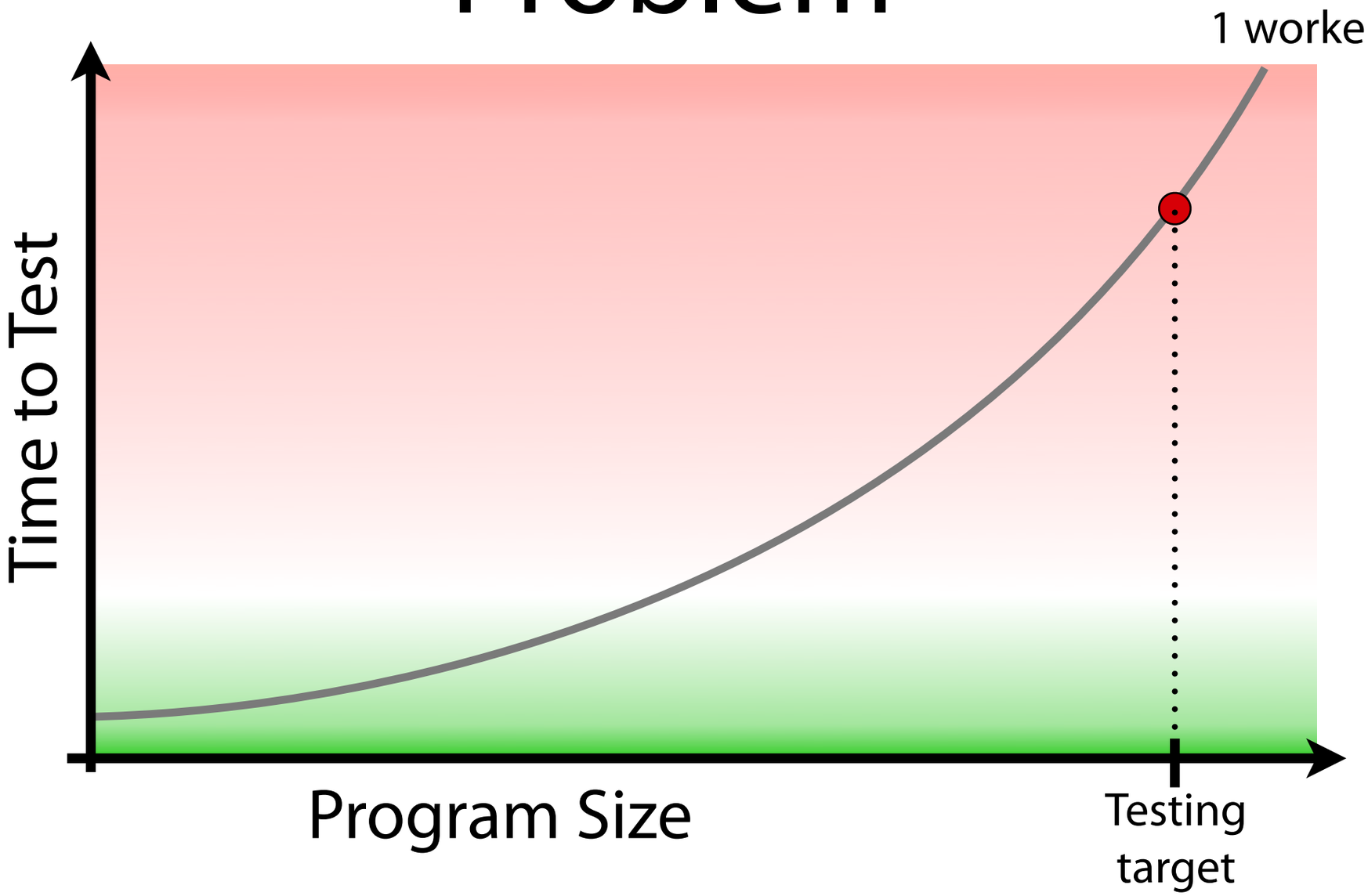


Key research problem:
Scalable parallel exploration

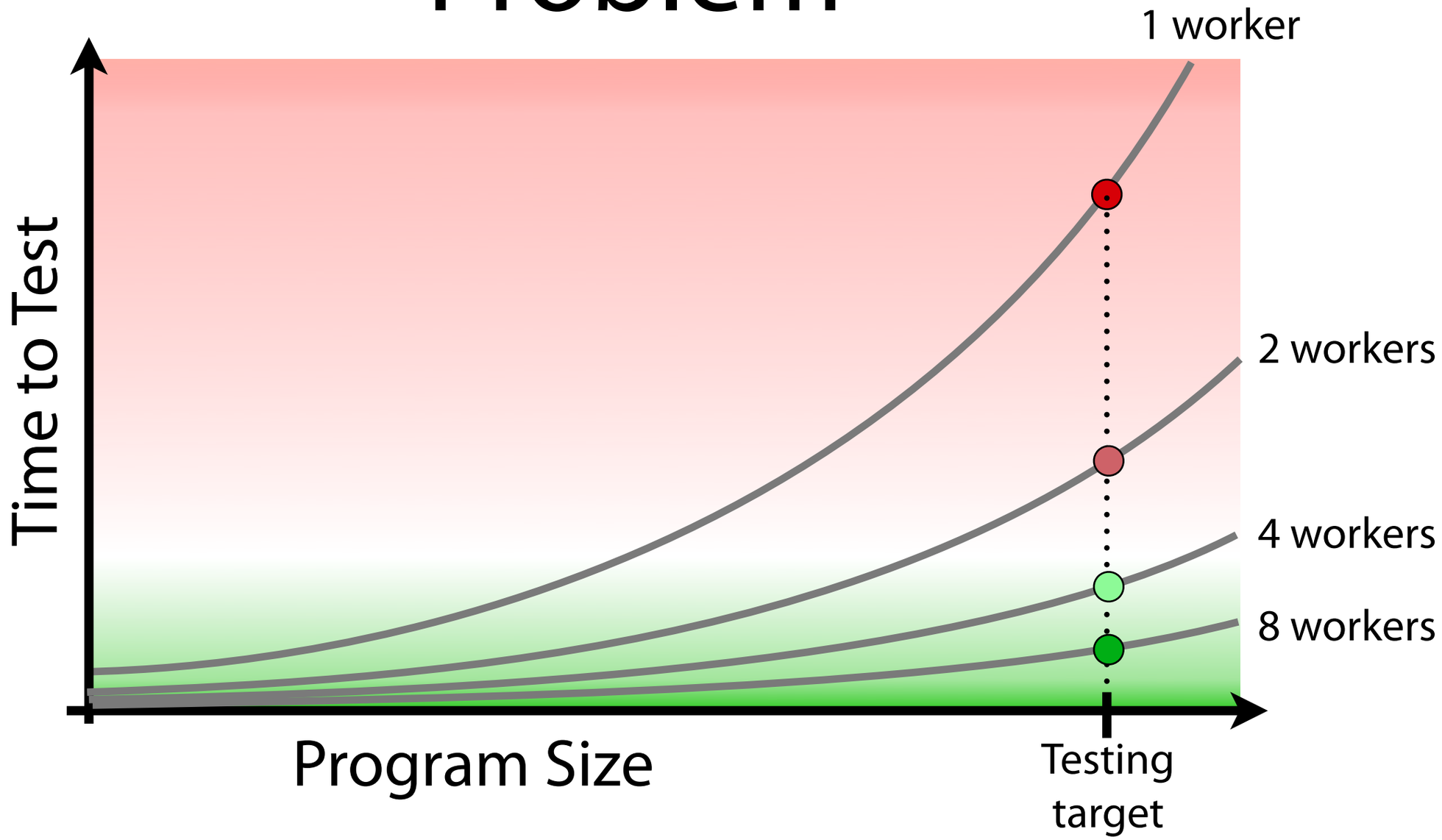
Linear Solution to Exponential Problem



Linear Solution to Exponential Problem



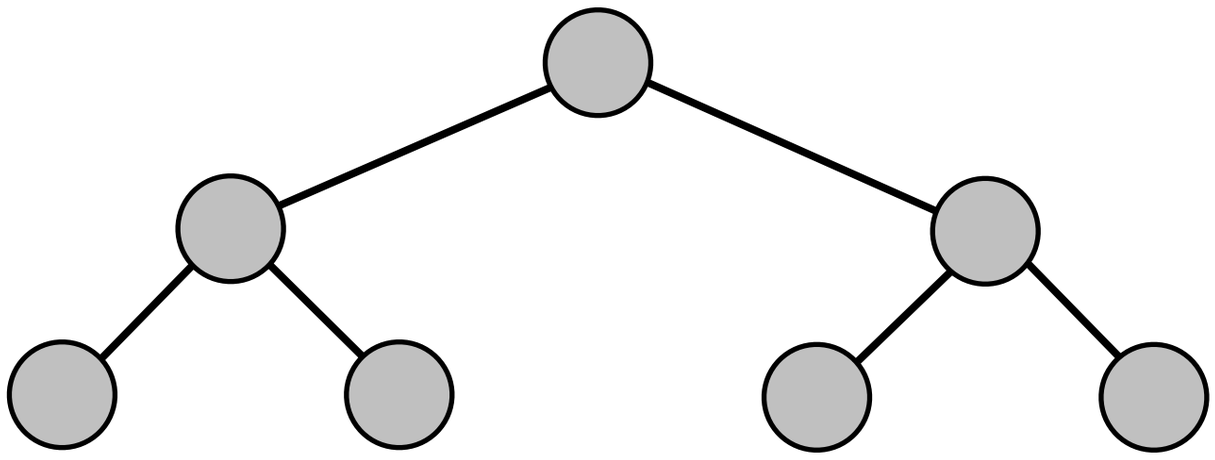
Linear Solution to Exponential Problem



Bring testing time down to practical values

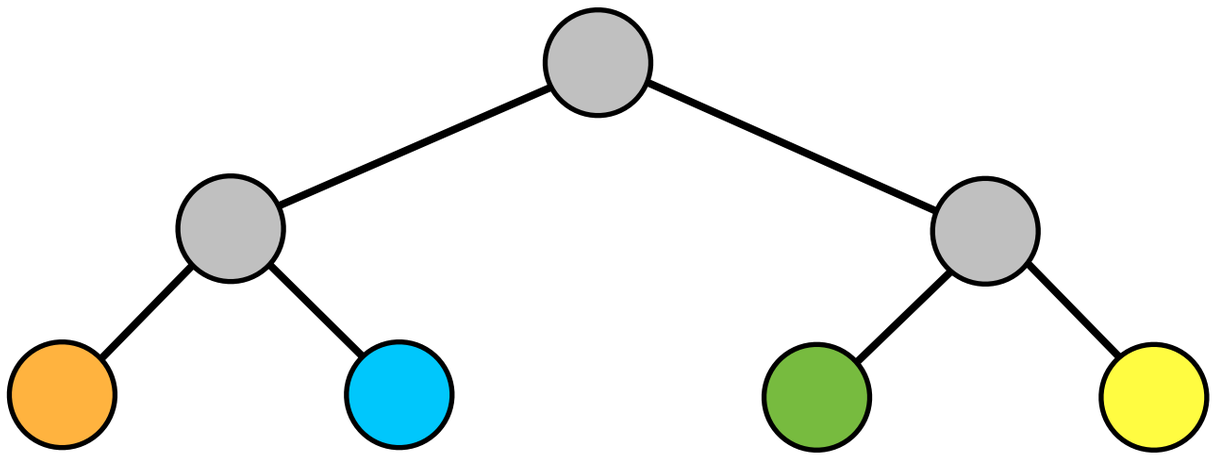
Throw Hardware at the Problem

Scalability Challenges



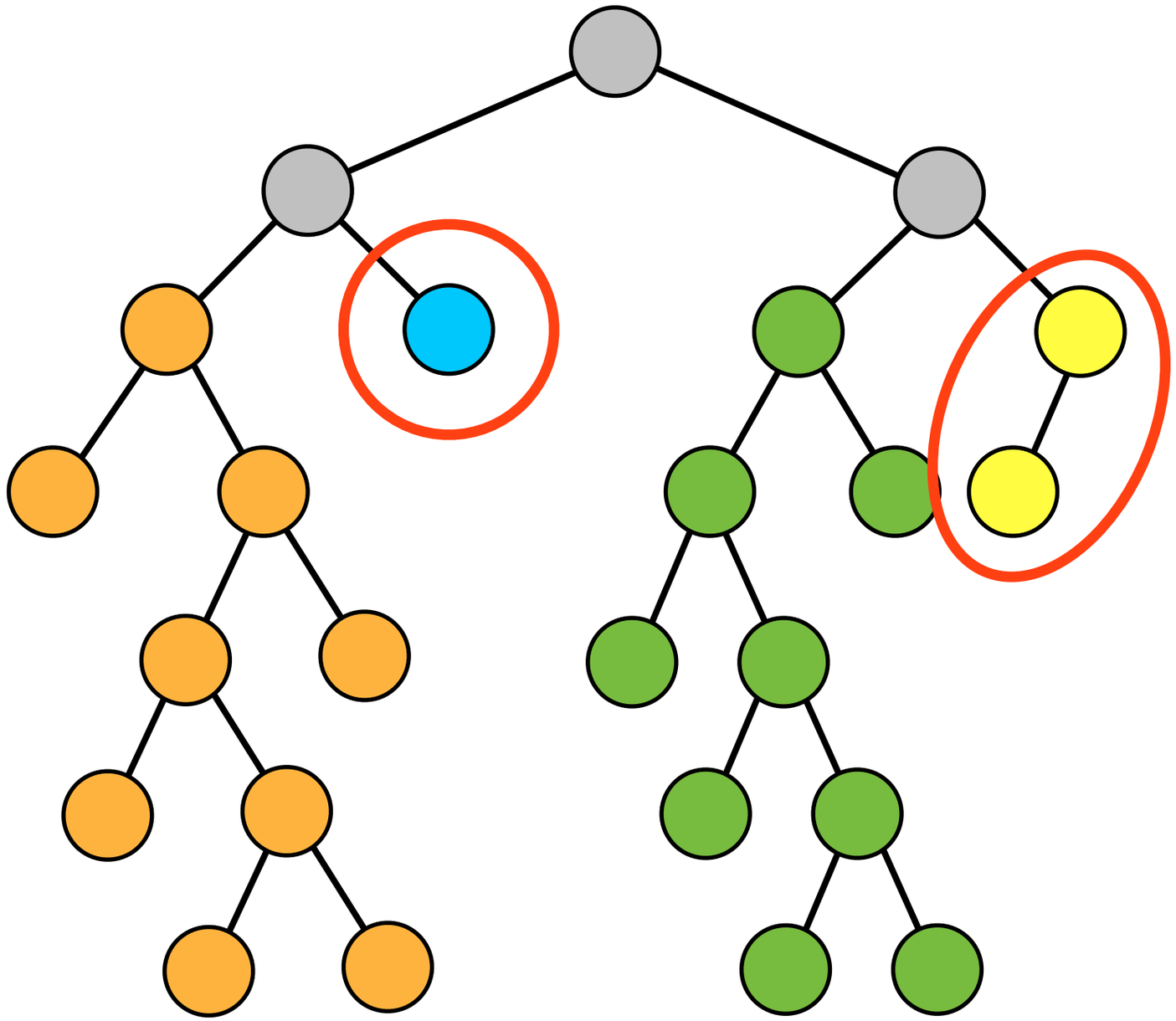
Tree structure not known a priori

Scalability Challenges

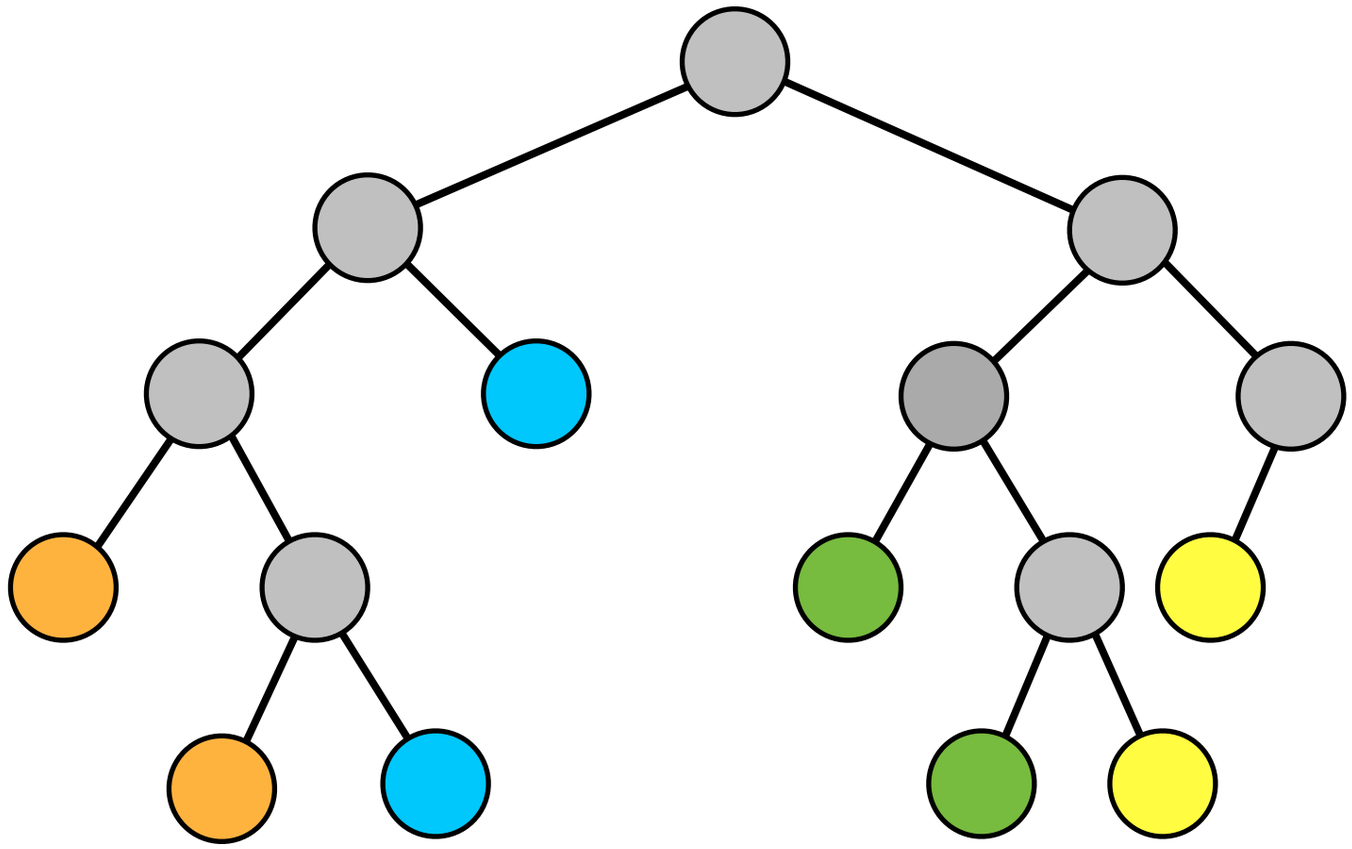


Static Allocation

Scalability Challenges

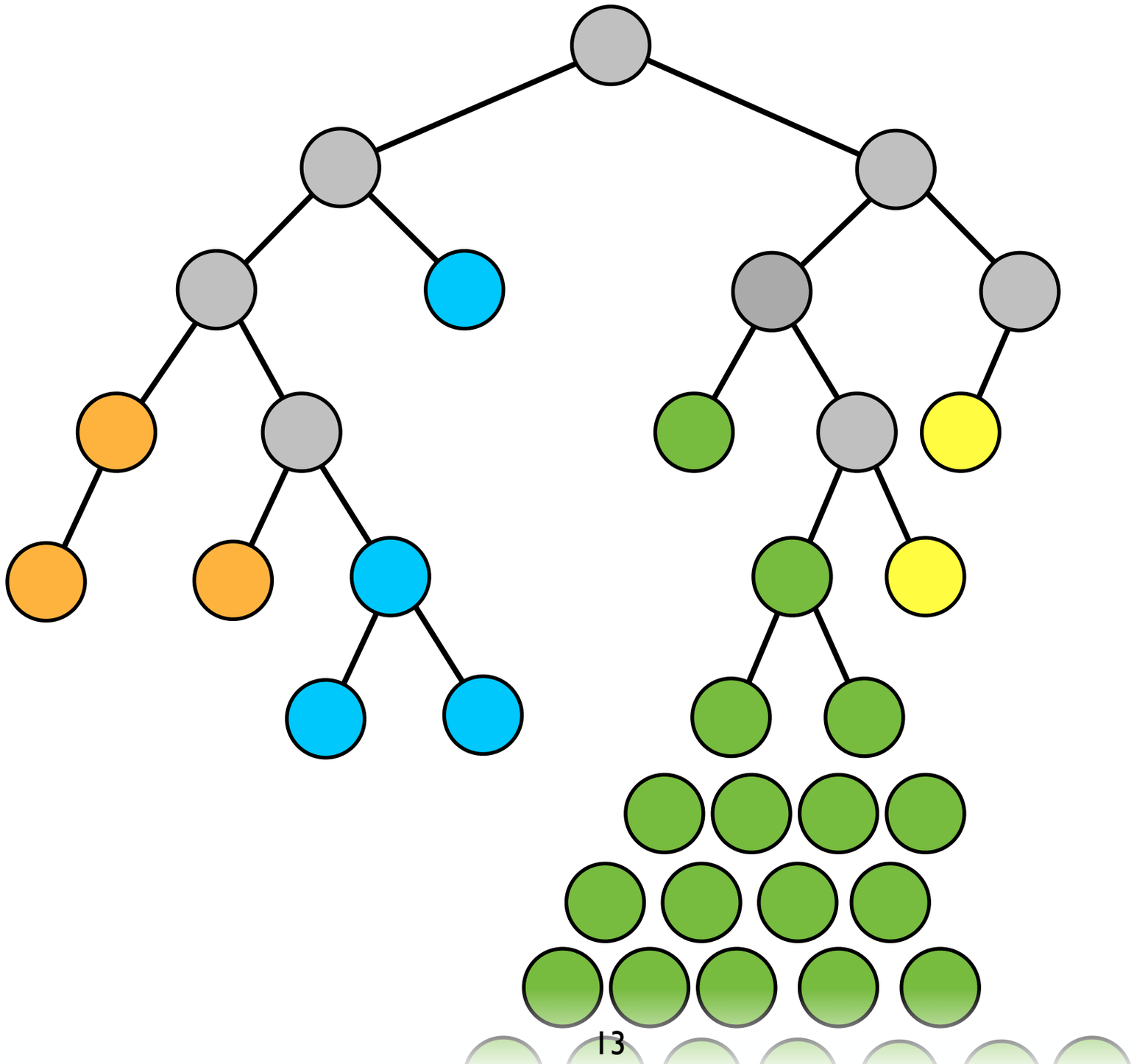


Scalability Challenges



Anticipate Allocation

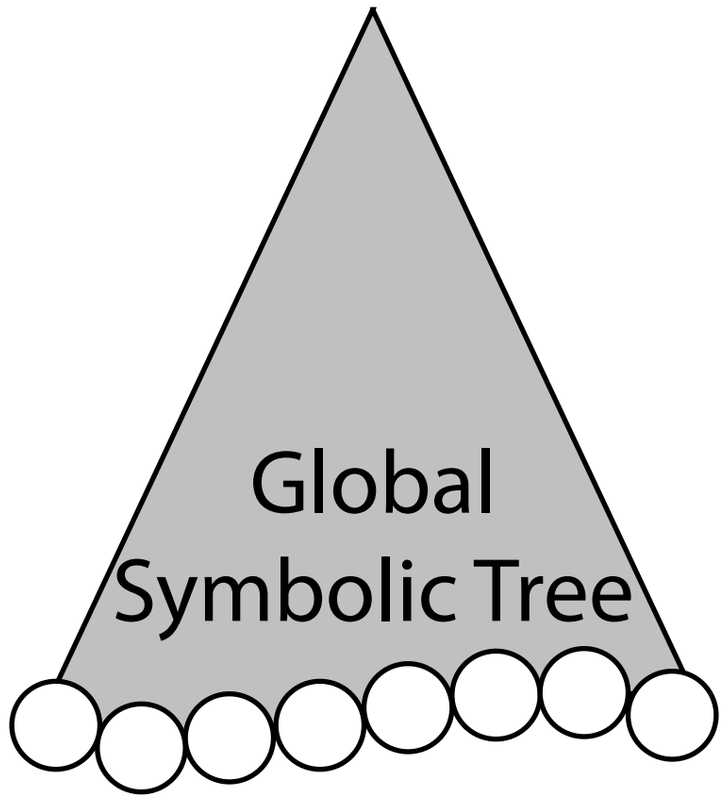
Scalability Challenges



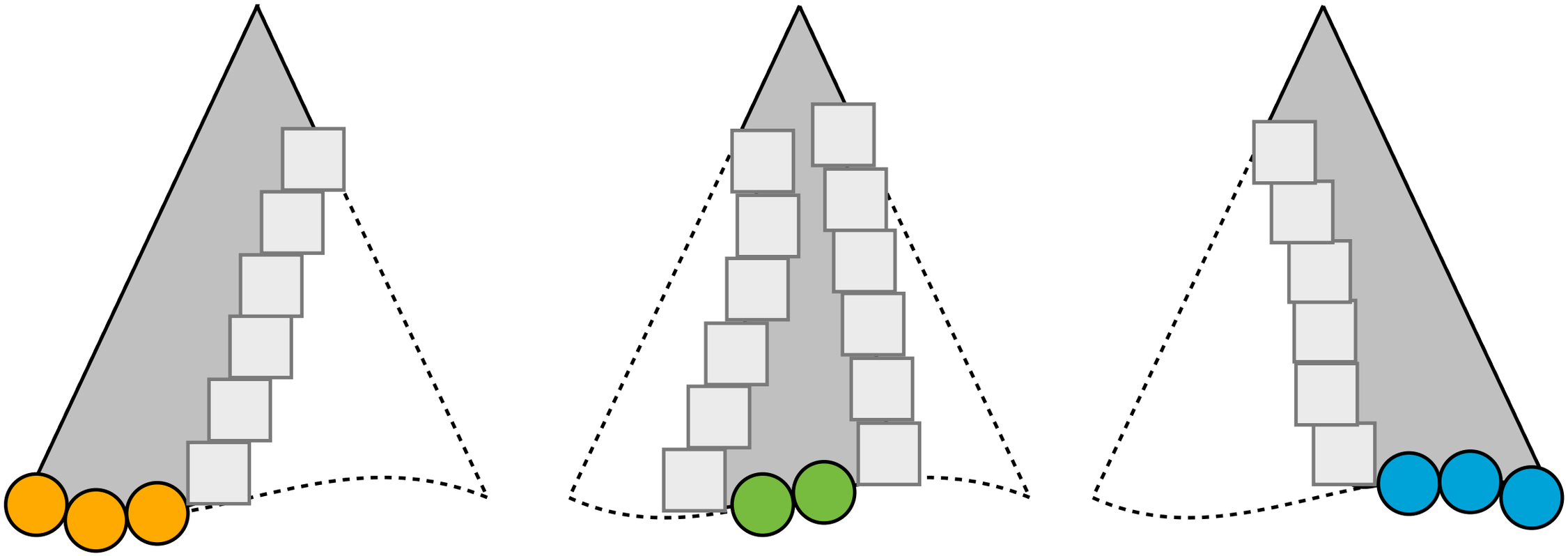
Outline

- **Scalable Parallel Symbolic Execution**
- POSIX Environment Model
- Evaluation

Cloud9 Architecture



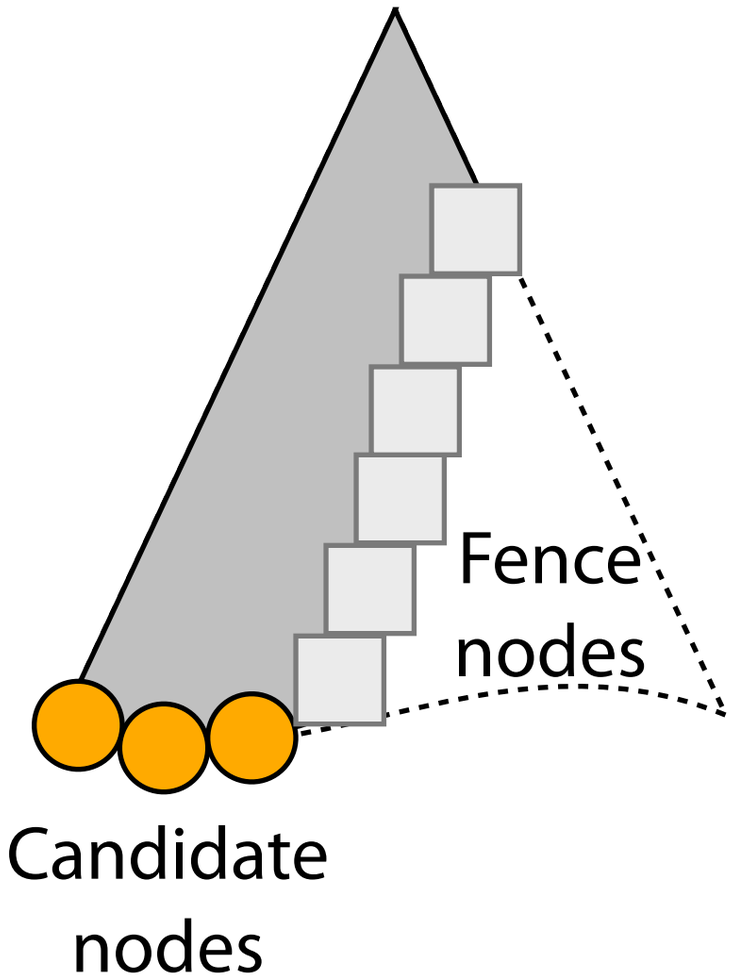
Cloud9 Architecture



W1's Local Tree W2's Local Tree W3's Local Tree

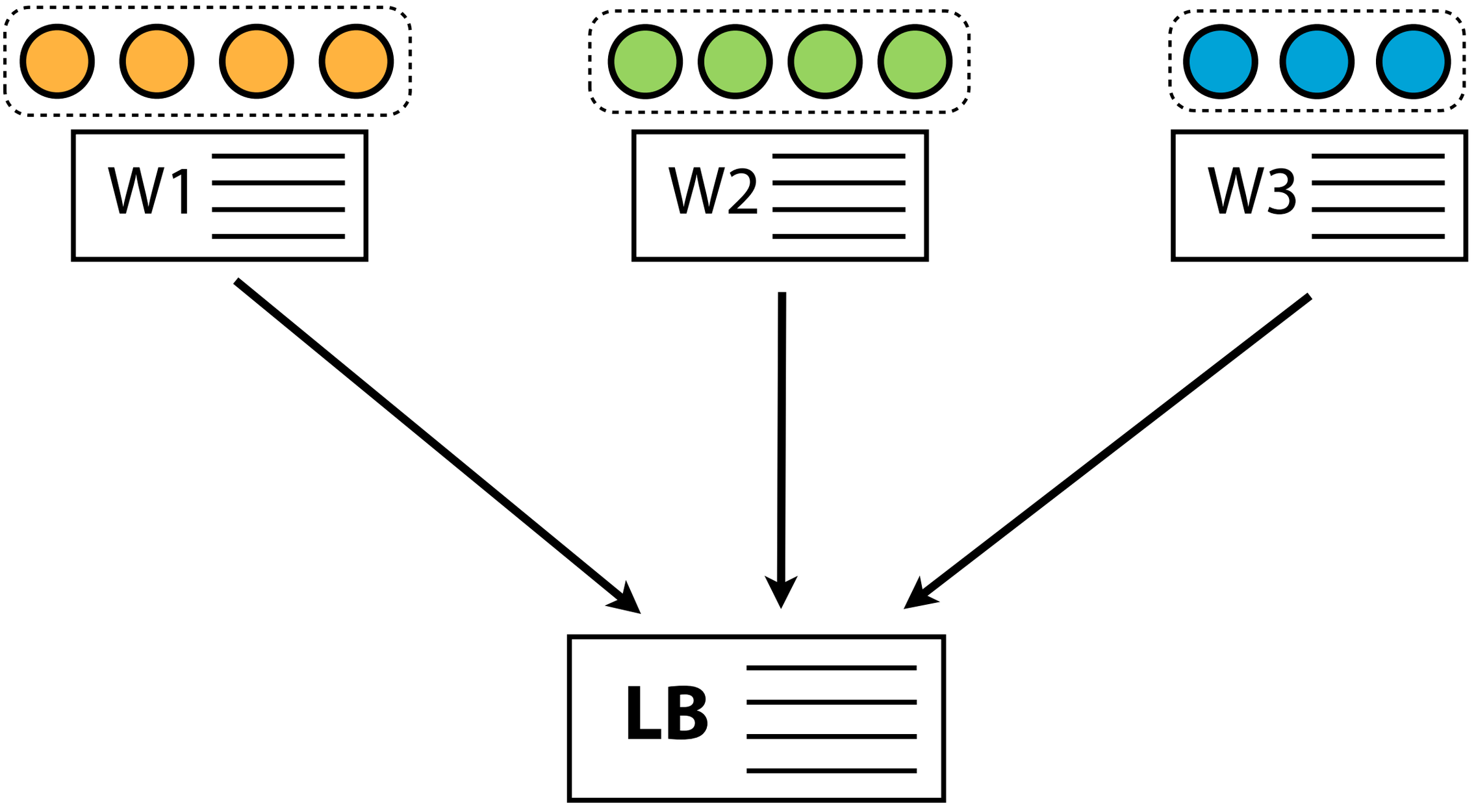
Each worker runs a local
sequential symbolic execution engine (KLEE)

Cloud9 Architecture



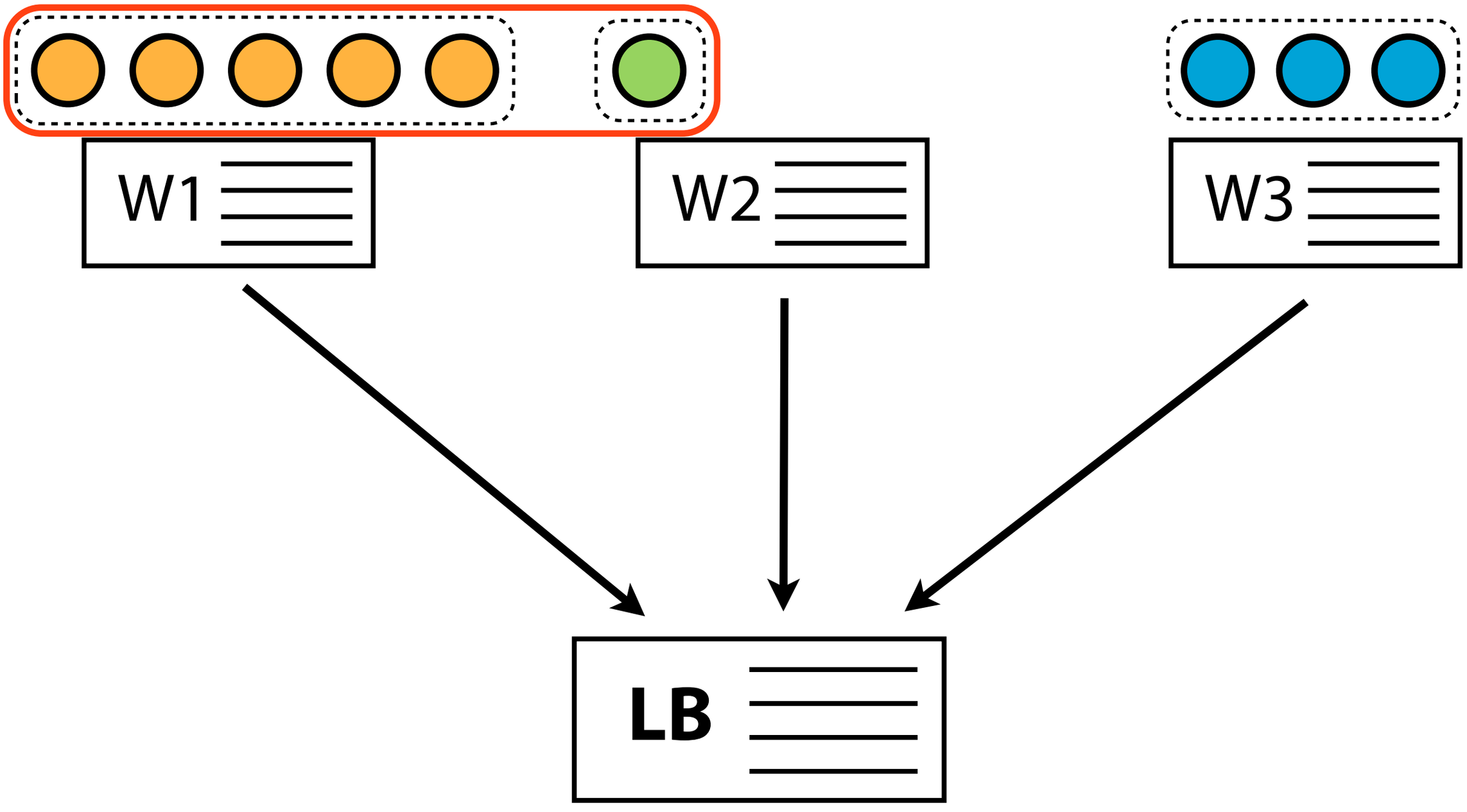
- Candidate nodes are selected for exploration
- Fence nodes bound the local tree

Load Balancing



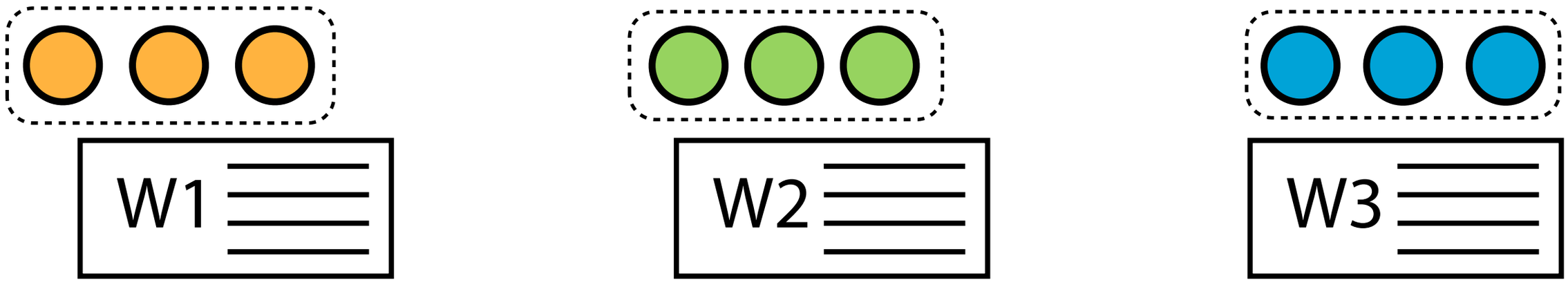
Hybrid distributed system:
centralized reports, P2P work transfer

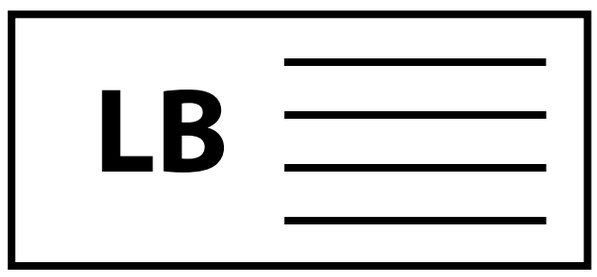
Load Balancing



Hybrid distributed system:
centralized reports, P2P work transfer

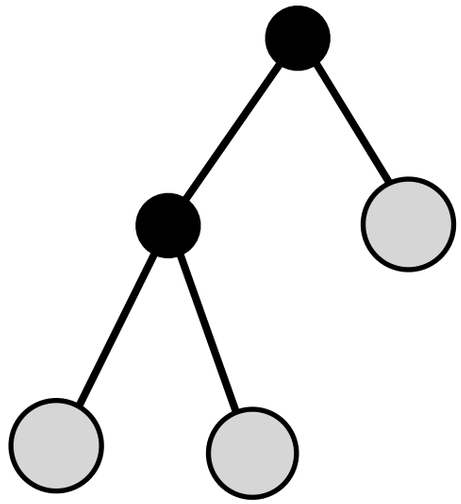
Load Balancing





Hybrid distributed system:
centralized reports, P2P work transfer

Work Transfer

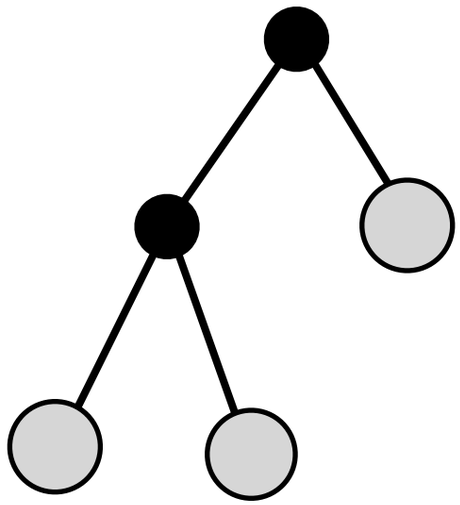


W1

Candidate

Fence

Work Transfer



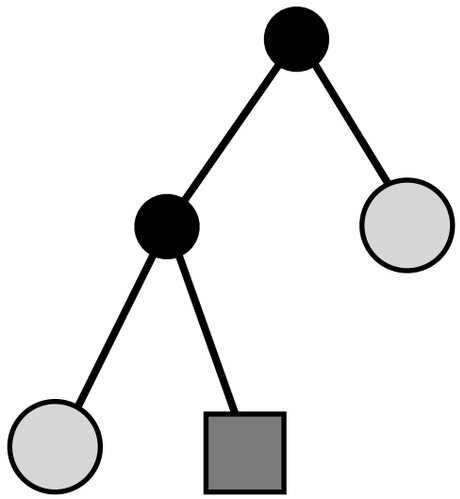
W1

W2

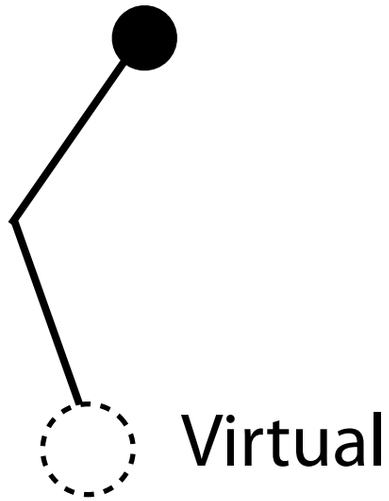
Candidate

Fence

Work Transfer



W1

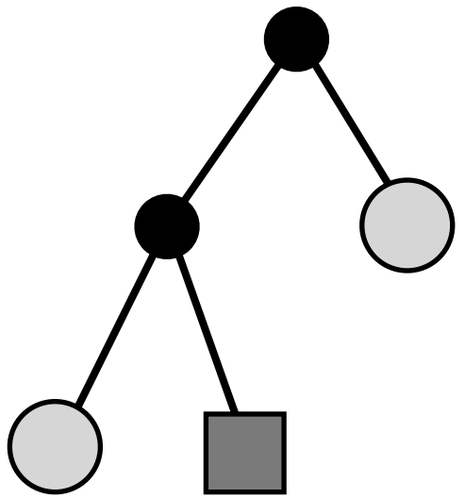


W2

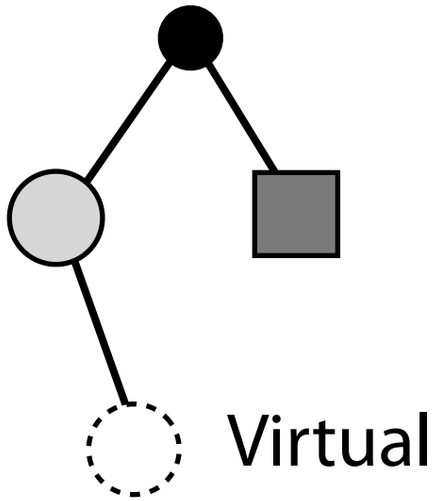
Candidate

Fence

Work Transfer



W1

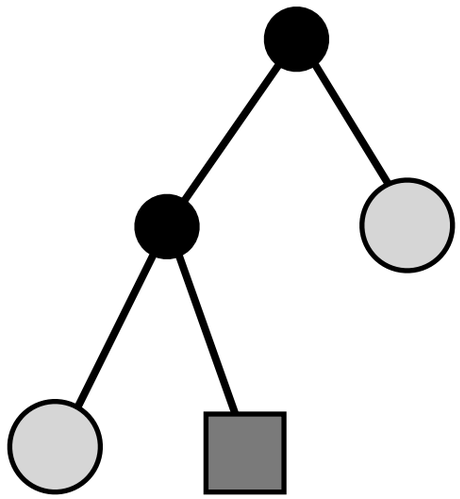


W2

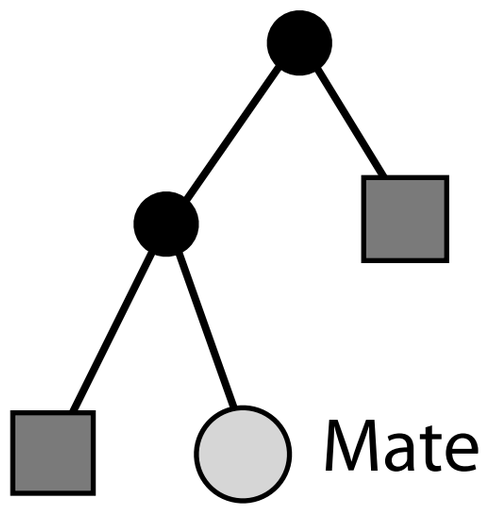
Candidate

Fence

Work Transfer



W1



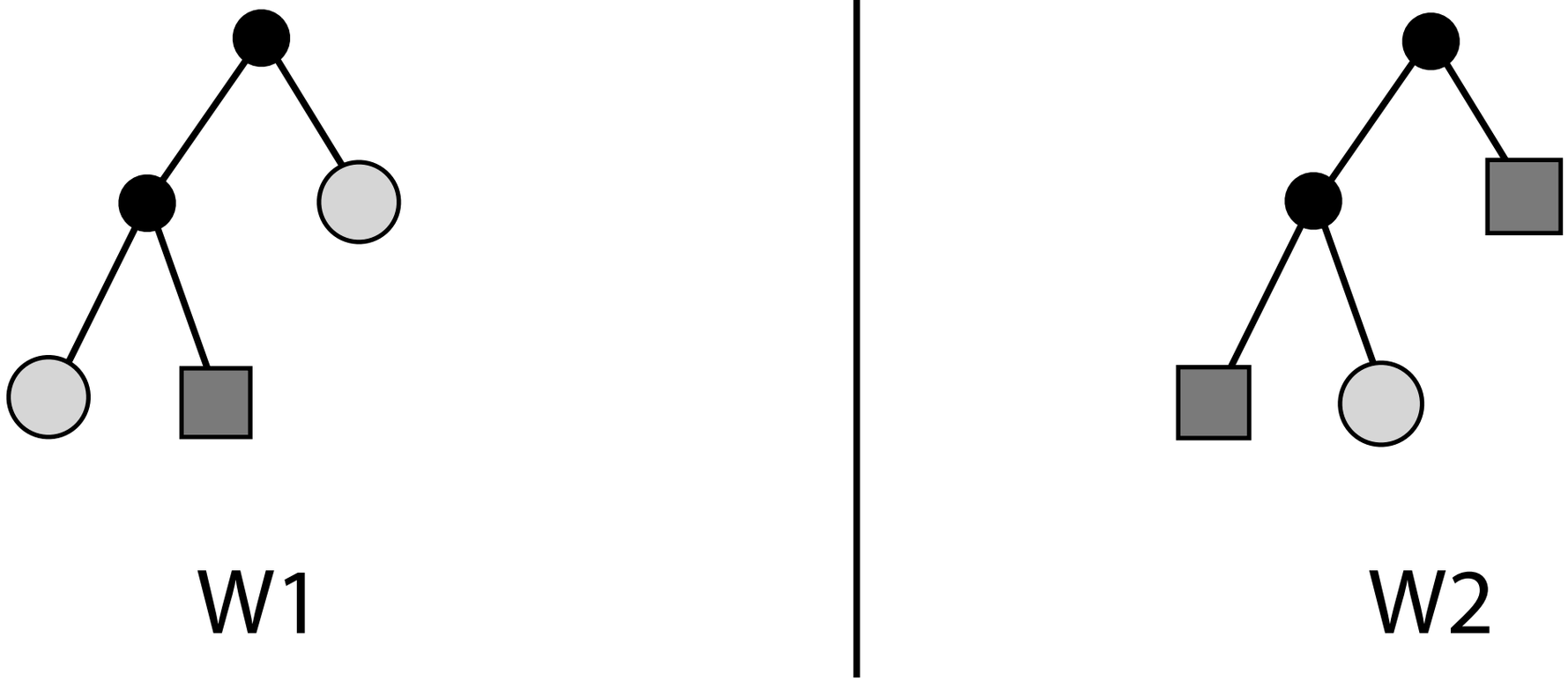
Materialized

W2

Candidate

Fence

Work Transfer



W1

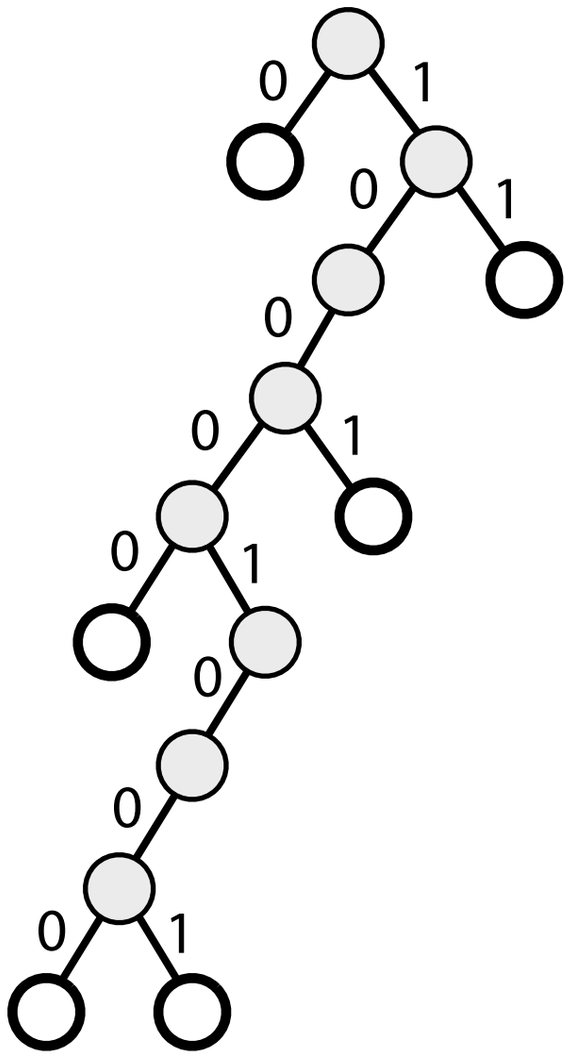
W2

Exploration disjointness + completeness

Candidate

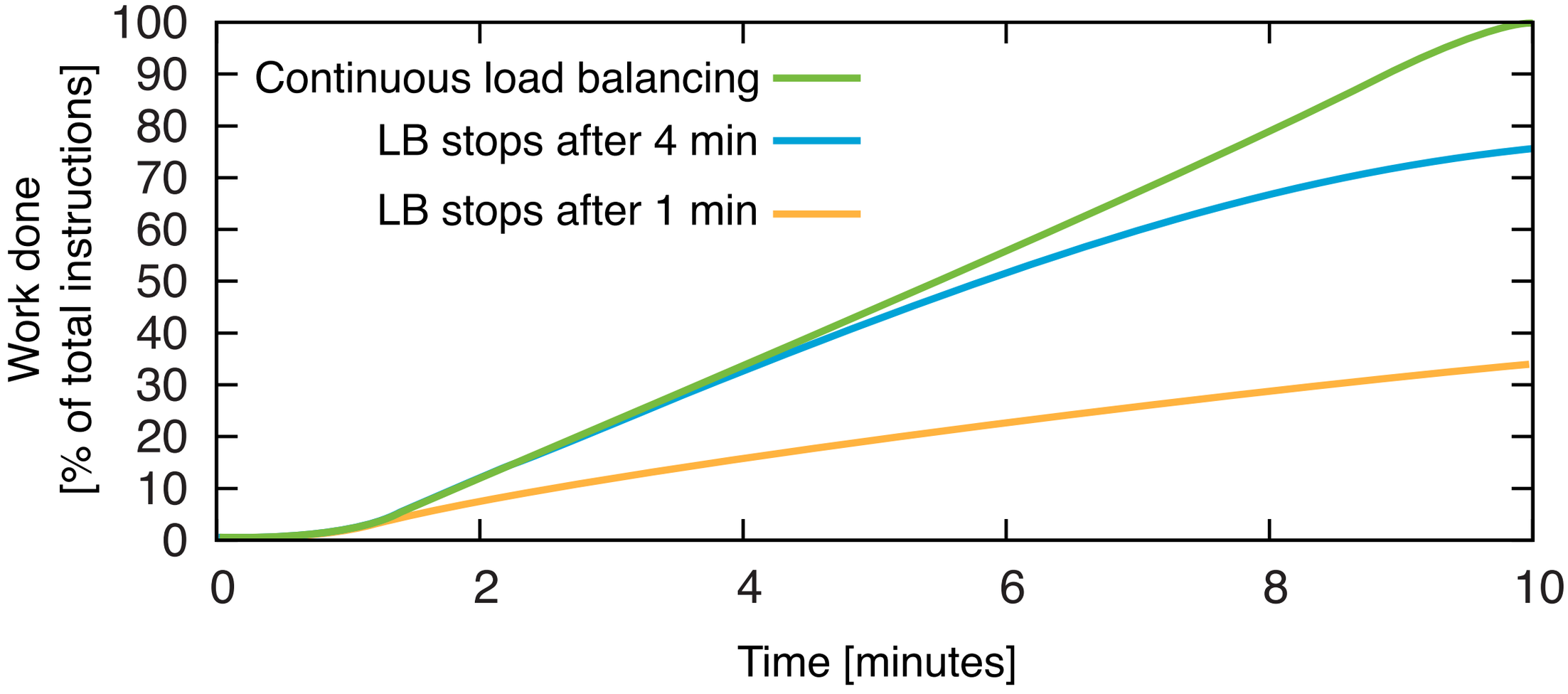
Fence

Path-based Encoding



- Nodes are encoded as paths in tree
  - *Compact binary representation*
  - *Two paths can share common prefix*
- Small encoding size
  - *For a tree of* 2^{100} *leaves, a path fits in* <128 *bits (*16 *bytes)*

Load Balancing in Practice



Load balancing necessary to ensure scalability

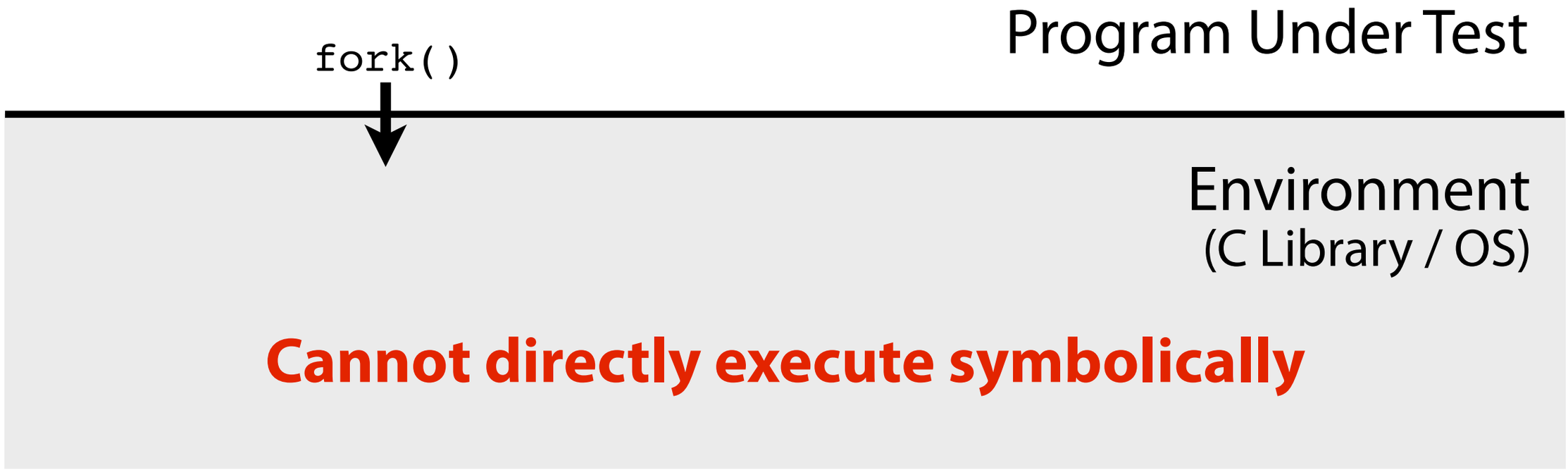
Outline

- Scalable Parallel Symbolic Execution
- **POSIX Environment Model**
- Evaluation

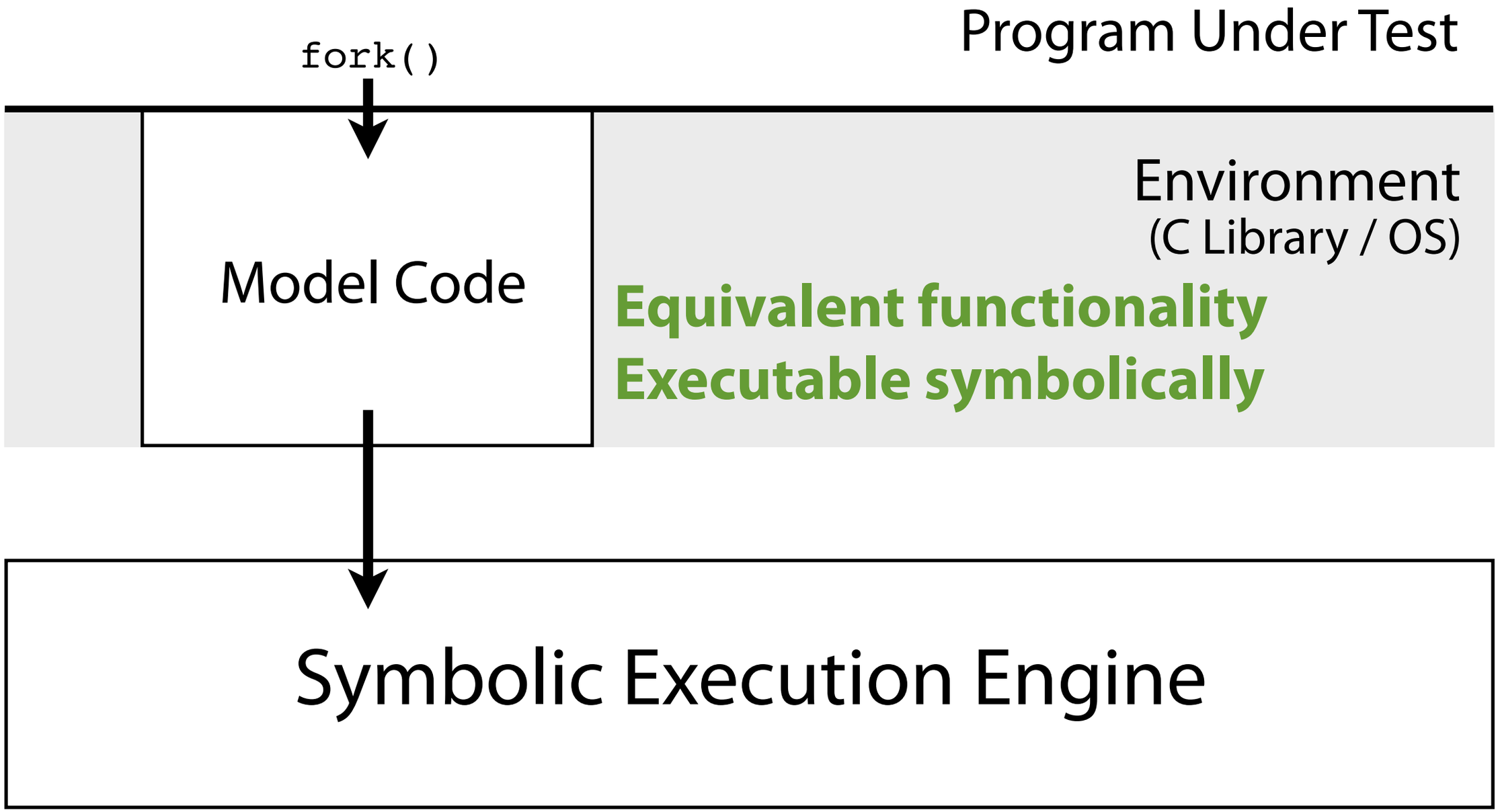
Calls into the Environment

```
if (fork() == 0) {
  ...
  if ((res = recv(sock, buff, size, 0)) > 0) {
    pthread_mutex_lock(&mutex);
    memcpy(gBuff, buff, res);
    pthread_mutex_unlock(&mutex);
  }
  ...
} else {
  ...
  pid_t pid = wait(&stat);
  ...
}
```

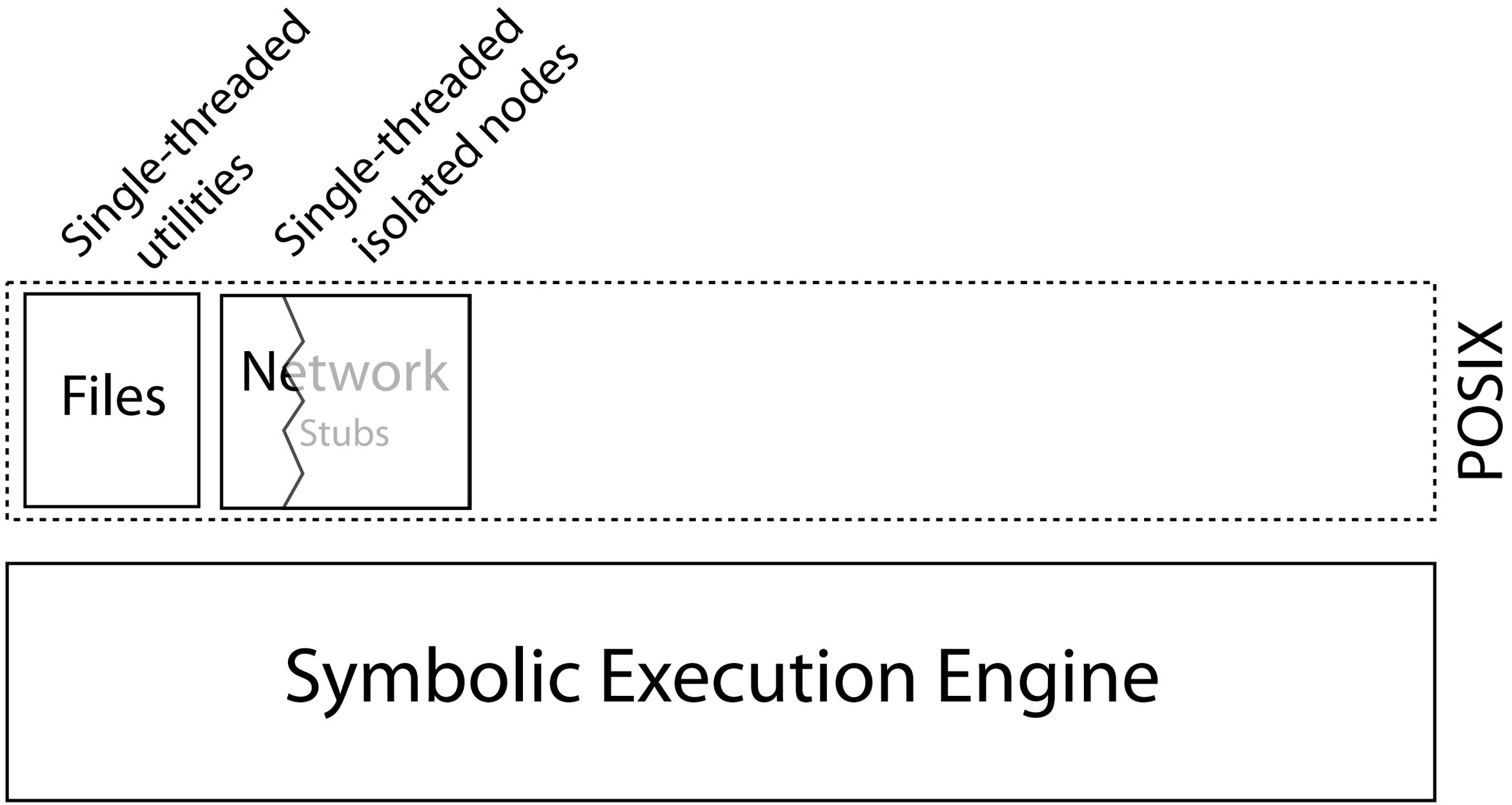
Environment Model



Environment Model



Starting Point



POSIX Environment Model

Single-threaded utilities

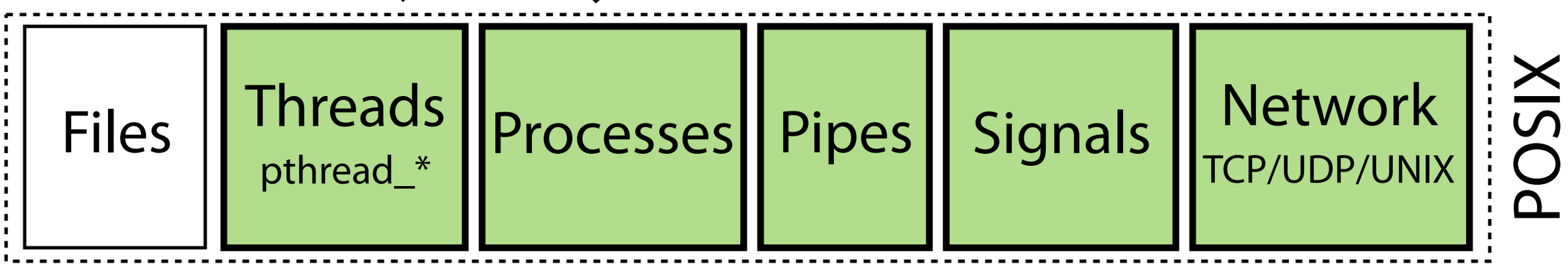
Multi-threaded programs

Distributed systems

Message passing

Asynchronous events, IPC

Servers and clients



Symbolic Execution Engine

Key Changes in Symbolic Execution

Multithreading and Scheduling

- Deterministic or symbolic scheduling
- Non-preemptive execution model

Address Space Isolation

- Copy on Write (CoW) between processes
- *CoW domains* for memory sharing

Symbolic Engine System Calls

- Symbolic engine support needed for threads/processes
  1. *Thread/process lifecycle*
  2. *Synchronization*
  3. *Shared memory*

Symbolic Engine System Calls	

1	`thread_create`
	`thread_terminate`
	`process_fork`
	`process_terminate`
	`get_context`

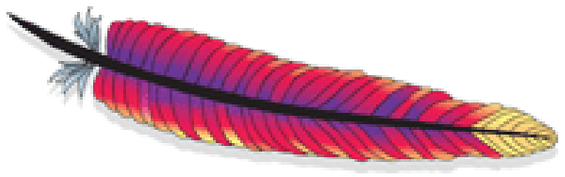
2	`thread_preempt`
	`thread_sleep`
	`thread_notify`
	`get_wait_list`

3	`make_shared`

Outline

- Scalable Parallel Symbolic Execution
- POSIX Environment Model
- **Evaluation**

Testing Real-World Software



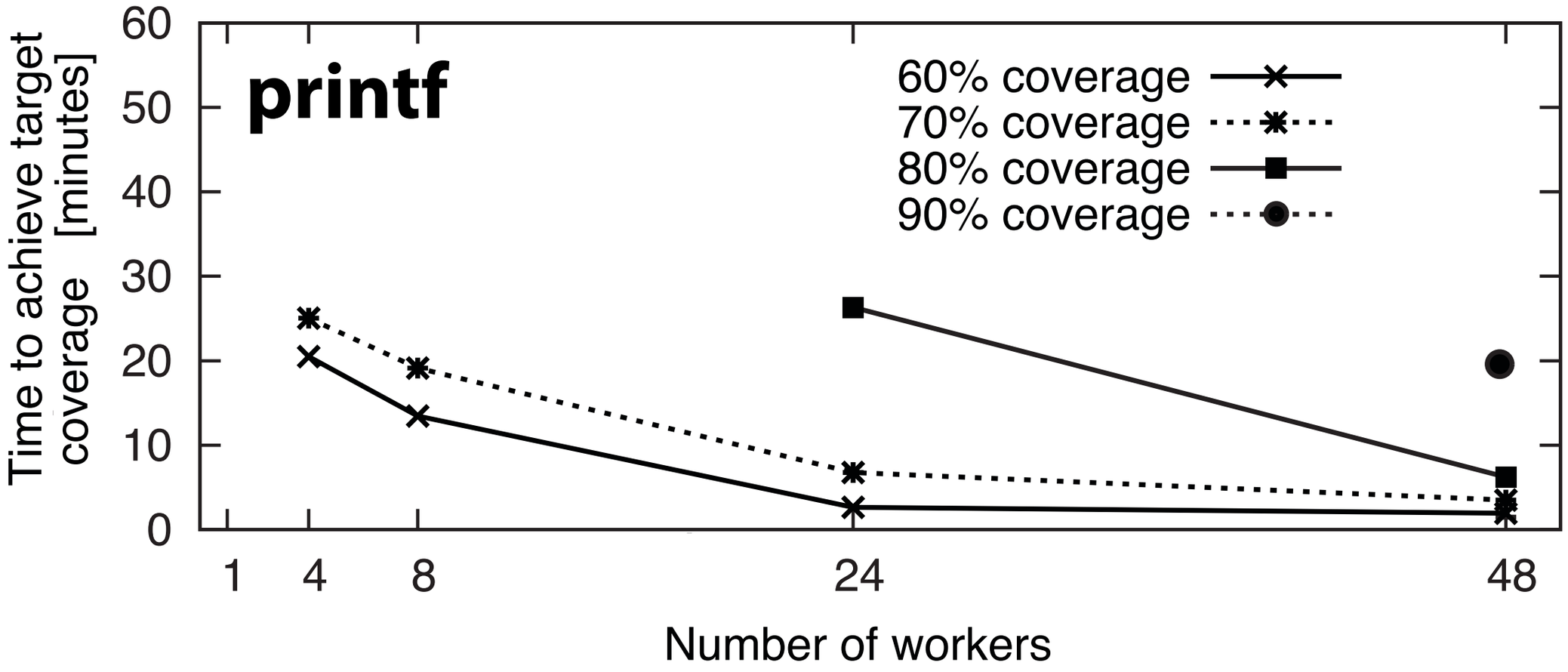
Apache

Memcached

GNU Coreutils

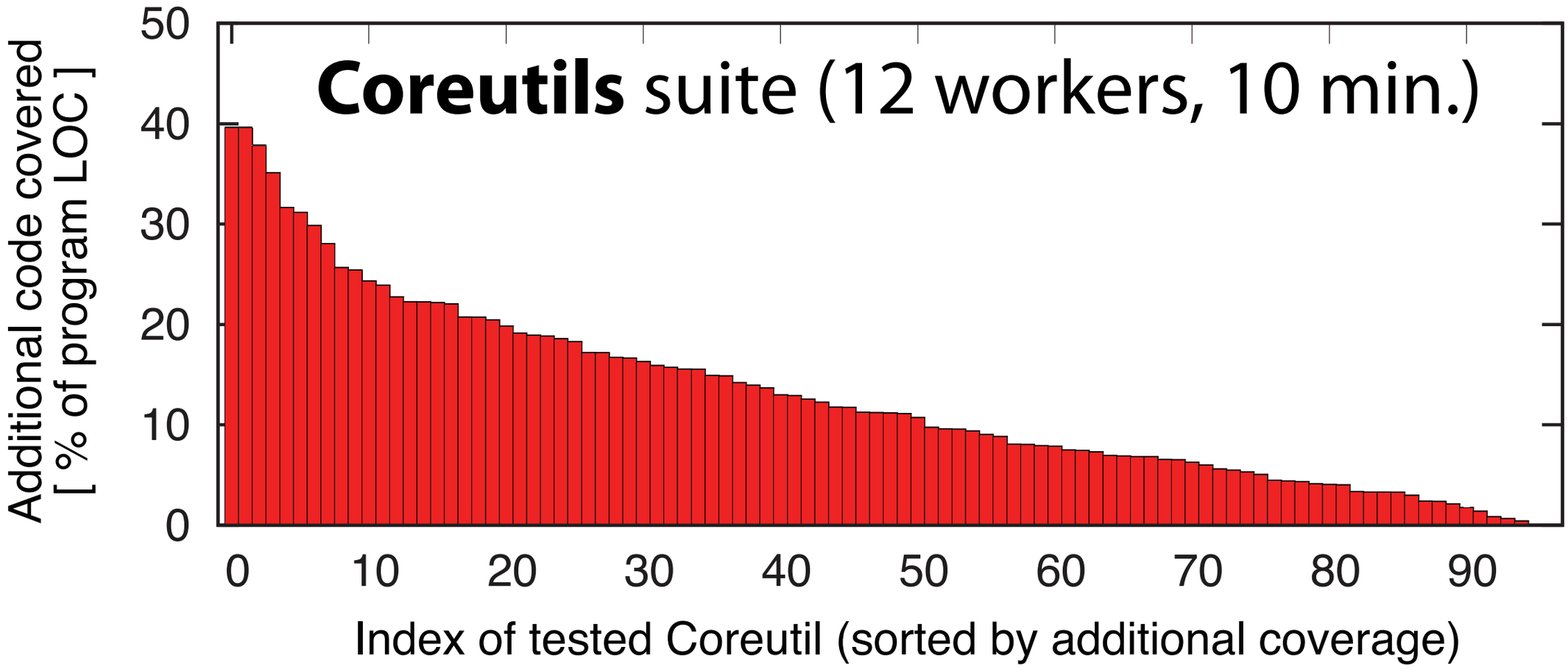
grabs those URLs

Time to Reach Target Coverage



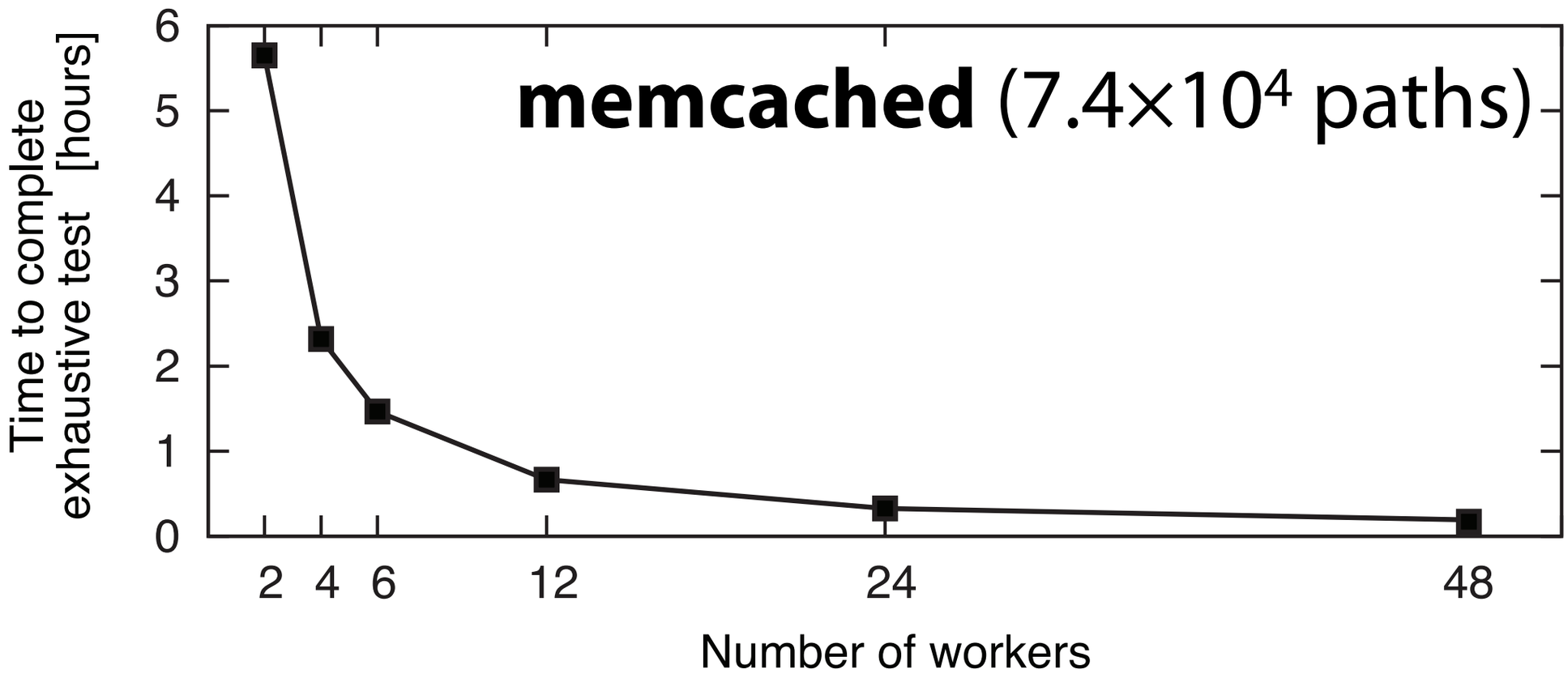
Faster time-to-cover, higher coverage values

Increase in Code Coverage



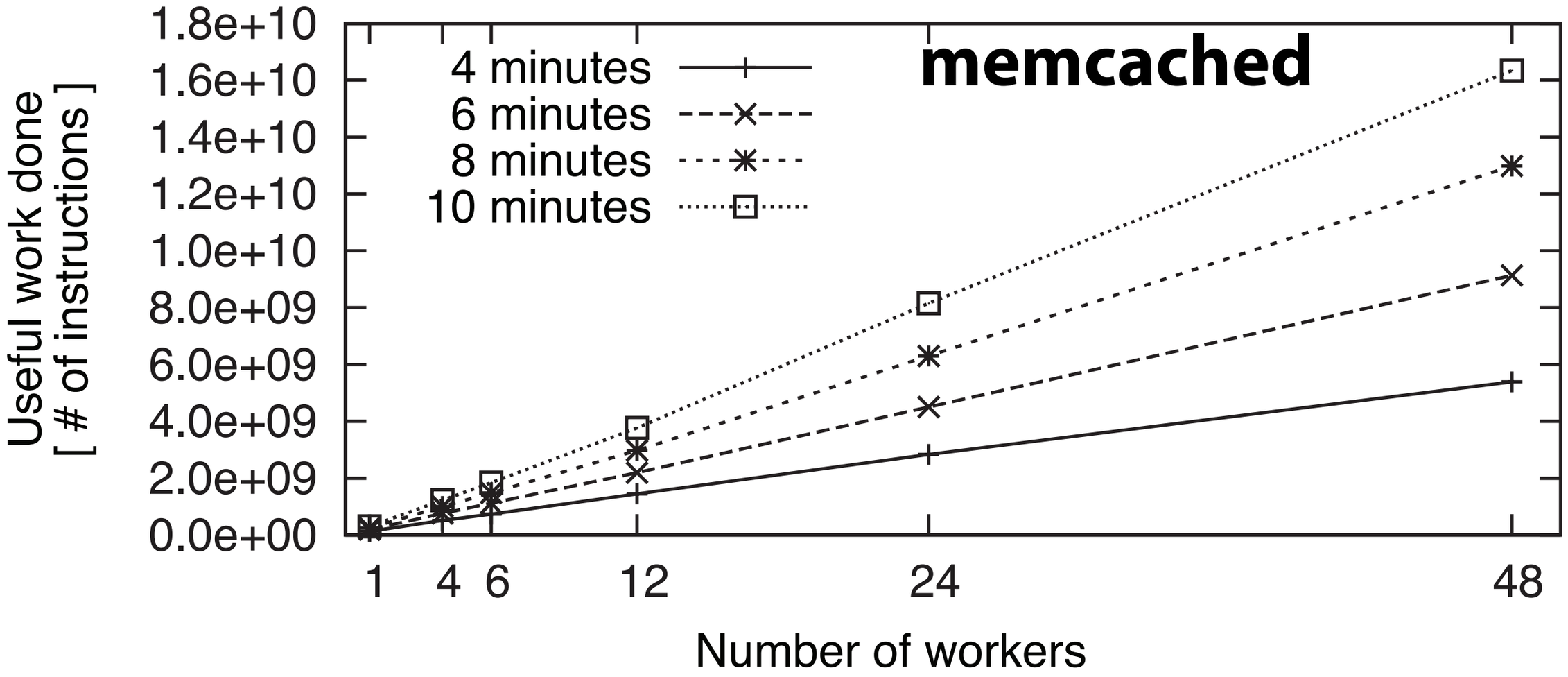
Consistent code coverage increase

Exhaustive Exploration



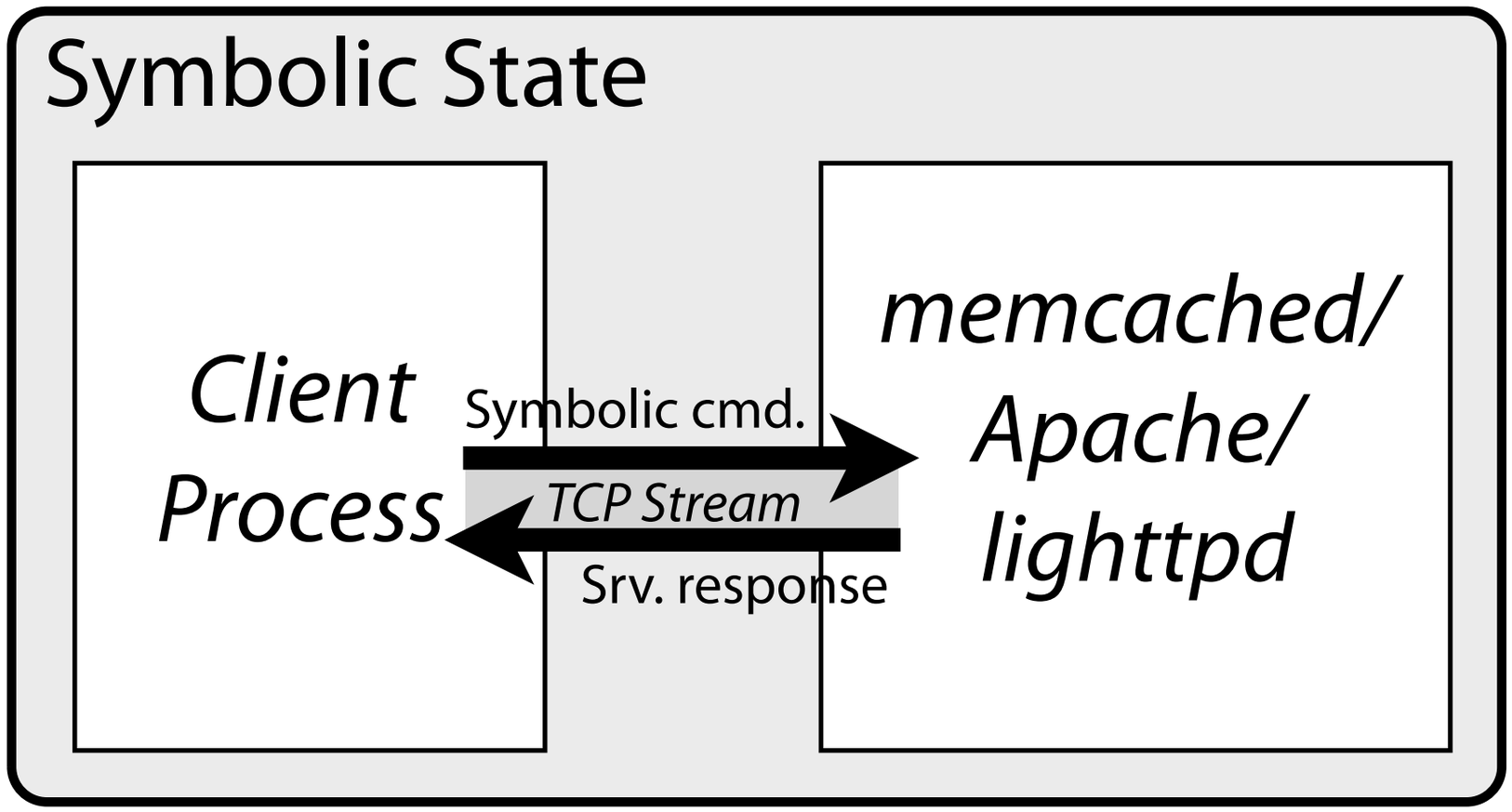
Scalability of exhaustive path exploration

Instruction Throughput



Linear scalability with number of workers

Experimental Setup



Execute the “whole world” symbolically

Symbolic Test Cases

- Easy-to-use API for developers to write symbolic test cases
- Basic symbolic memory support
- POSIX extensions for environment control
  - *Network conditions, fault injection, symbolic scheduler*

Symbolic Test Cases

Testing HTTP header extension

```
make_symbolic(hdrData);
// Append symbolic header to request
strcat(req, “X-NewExtension: “);
strcat(req, hdrData);

// Enable fault injection on socket
ioctl(ssock, SIO_FAULT_INJ, RD | WR);
// Symbolic stream fragmentation
ioctl(ssock, SIO_PKT_FRAGMENT, RD);
```

Conclusions

- Parallel symbolic execution
  - *Linear scalability on commodity clusters*
- Full POSIX environment model
  - *Real-world systems testing*
- Use cases
  - *Increasing coverage*
  - *Exhaustive path exploration*
  - *Bug patch verification*