Declarative, Temporal, and Practical Programming with Capabilities

William Harris, Somesh Jha, Thomas Reps

Jonathan Anderson, Robert Watson

Paper in One Slide

- Capsicum supports secure programming, but secure programming is still hard

- CapWeave instruments programs to be secure on Capsicum

Talk Outline

1. Why use Capsicum? (USENIX Security '10)
2. Why use CapWeave?
3. How does CapWeave work?
4. How well does CapWeave work?

Talk Outline

1. Why use Capsicum? (USENIX Security '10)

A. A Capsicum process can sandbox itself by invoking a few custom system primitives

gzip

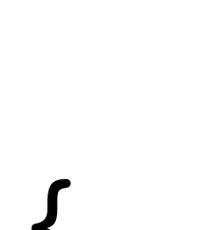
```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```

gzip

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```

gzip

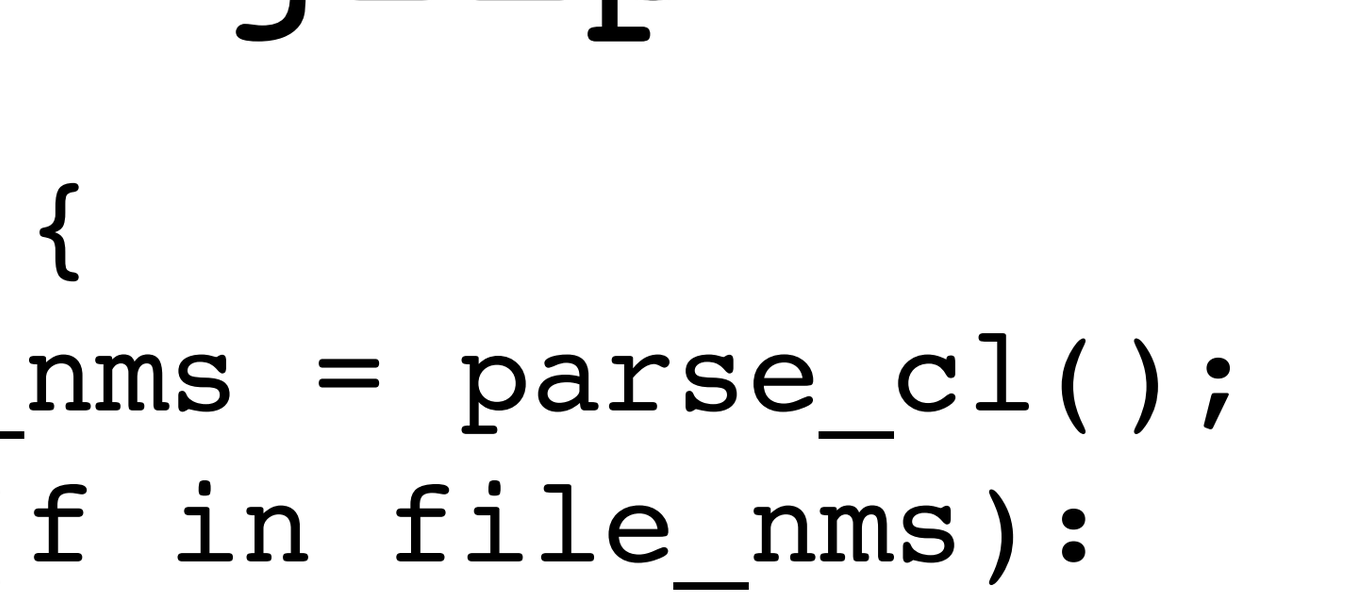
```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```



http://evil.com

gzip

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```



A simple `gzip` policy

- When `gzip` calls `open2()` at `L0`, it should be able to open descriptors

- When `gzip` calls `compress()` at `L1`, it should not be able to open descriptors

Capsicum's AMB

A Capsicum process can open descriptors if and only if it has *ambient authority* (AMB)

Rules for Capsicum's AMB

Rules for Capsicum's AMB

1. When a process is created,
   it has the AMB value of its parent

Rules for Capsicum's AMB

1. When a process is created,
   it has the AMB value of its parent

2. After a process calls `cap_enter()`,
   it never has AMB

A simple `gzip` policy using Capsicum's AMB

- When `gzip` calls `open2()` at `L0`, it should be able to open descriptors
- When `gzip` calls `compress()` at `L1`, it should not able to open descriptors

A simple `gzip` policy using Capsicum's AMB

- When `gzip` calls `open2()` at `L0`, it should have AMB
- When `gzip` calls `compress()` at `L1`, it should not have AMB

gzip using Capsicum's AMB

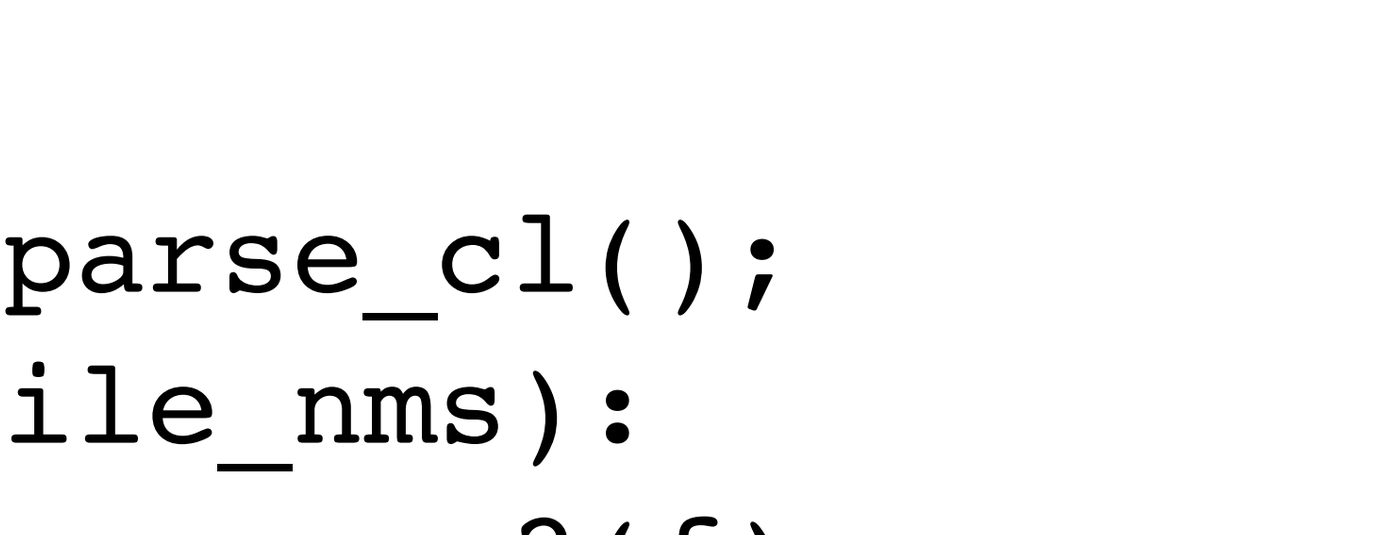
```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```

`gzip` using Capsicum's AMB

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```

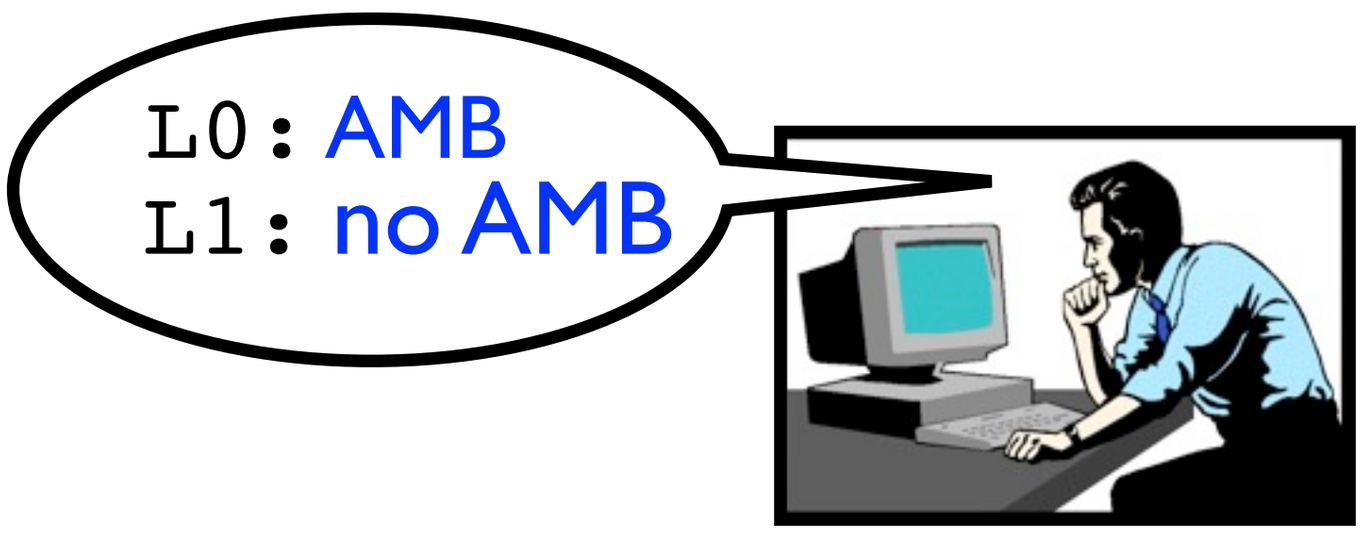
`gzip` using Capsicum's AMB

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```



`gzip` using Capsicum's AMB

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):                    cap_enter()
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```



gzip using Capsicum's AMB

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```

cap_enter()

L0: AMB
L1: no AMB

Talk Outline

1. Why use Capsicum? (USENIX Security '10)
2. Why use CapWeave?
3. How does CapWeave work?
4. How well does CapWeave work?

Talk Outline

2. Why use CapWeave?

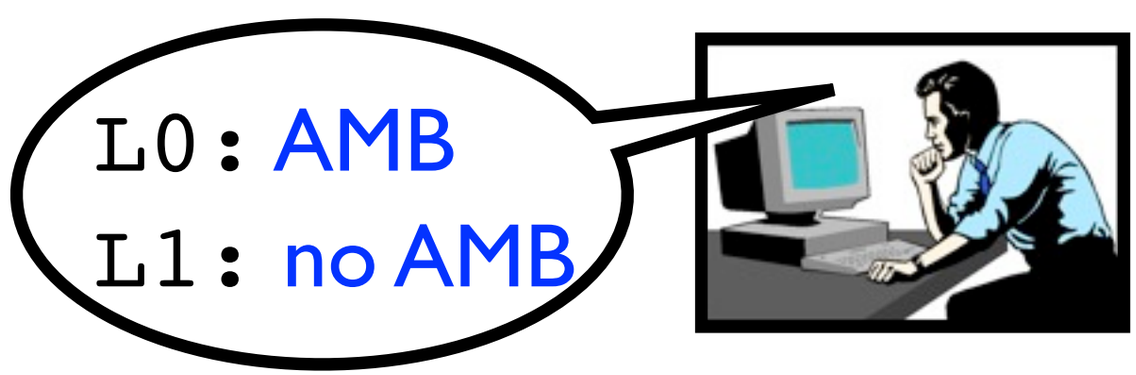
A. CapWeave bridges Capsicum's "semantic gap"

Capsicum Programming Challenges

1. Policies aren't explicit
2. Primitives have subtle temporal effects

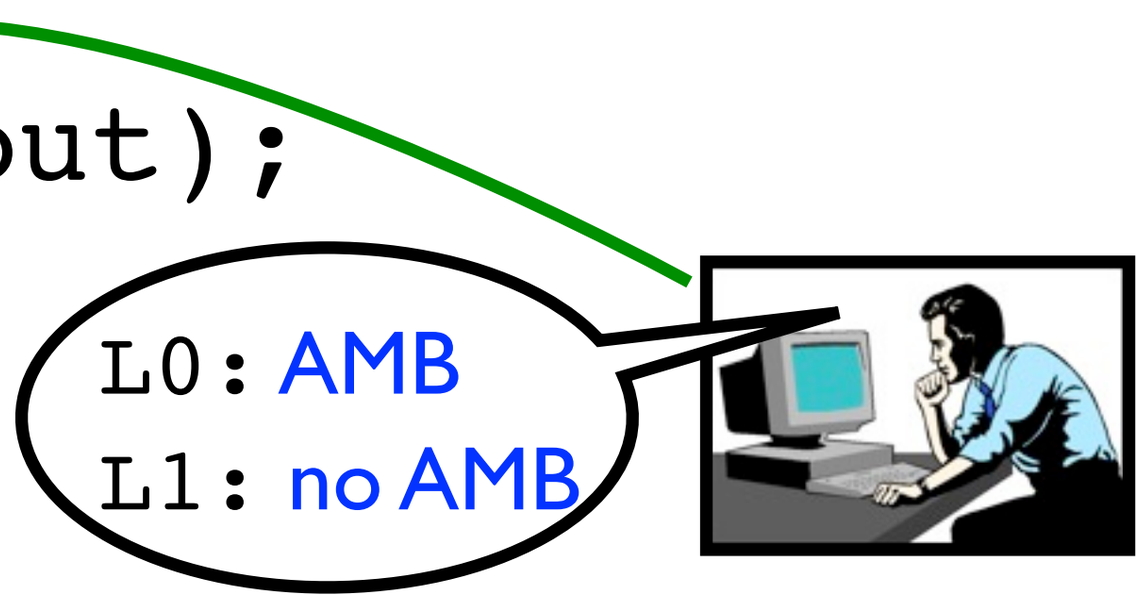
gzip
Programming Challenges

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);

L1: compress(in, out);
}
```



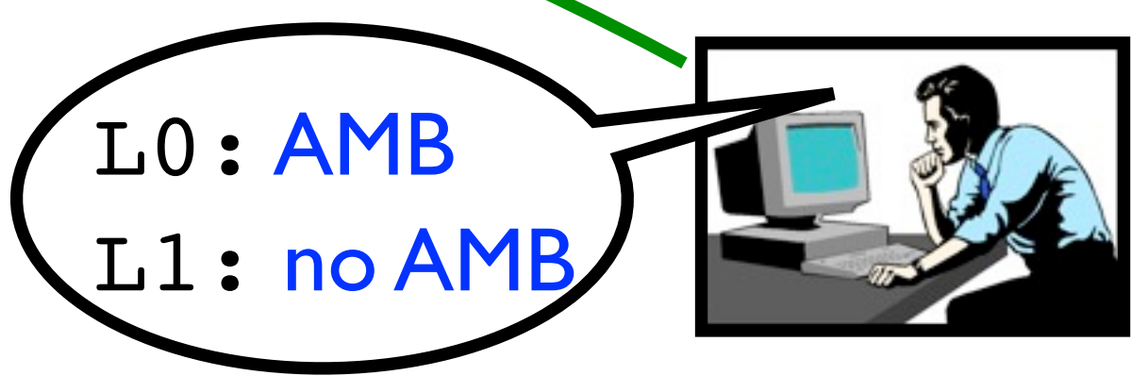
gzip
Programming Challenges

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
    cap_enter();
L1: compress(in, out);
}
```



gzip
Programming Challenges

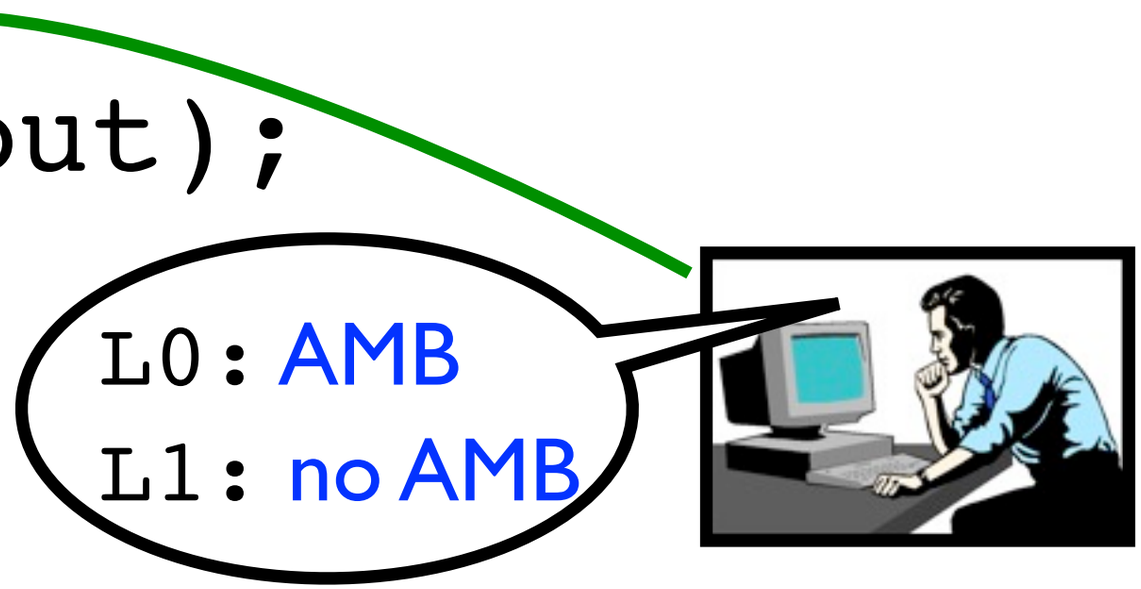
```
main() {
   file_nms = parse_cl();  AMB
   for (f in file_nms):
L0: (in, out) = open2(f);
    cap_enter();
L1: compress(in, out);
}
```



gzip
Programming Challenges

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
      cap_enter();
L1: compress(in, out);
}
```

AMB

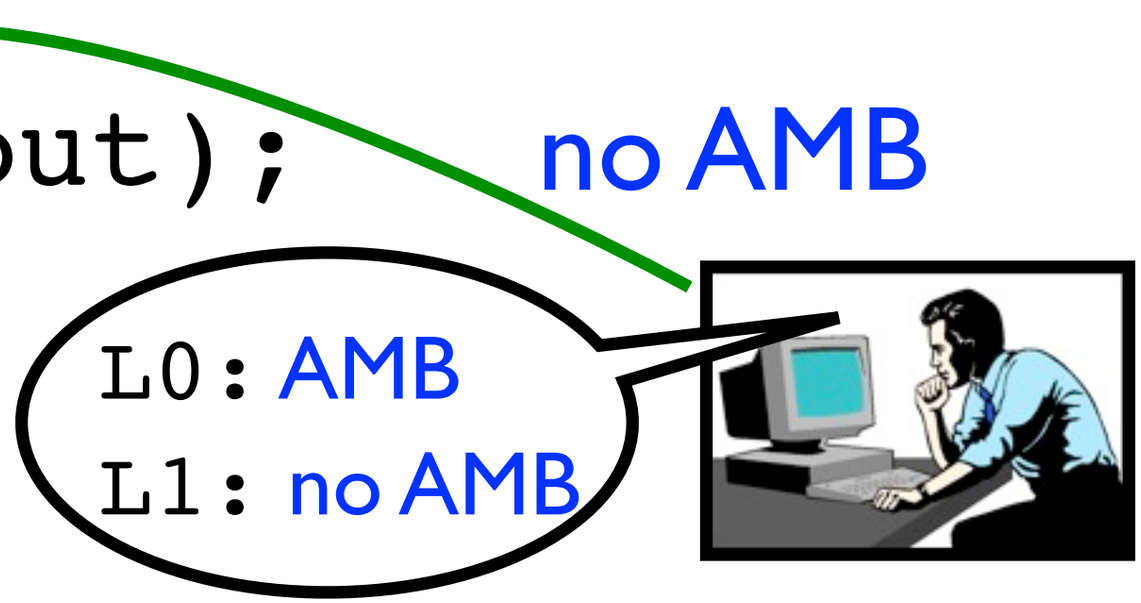


gzip
Programming Challenges

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);AMB
    cap_enter();
L1: compress(in, out);
}
```

gzip
Programming Challenges

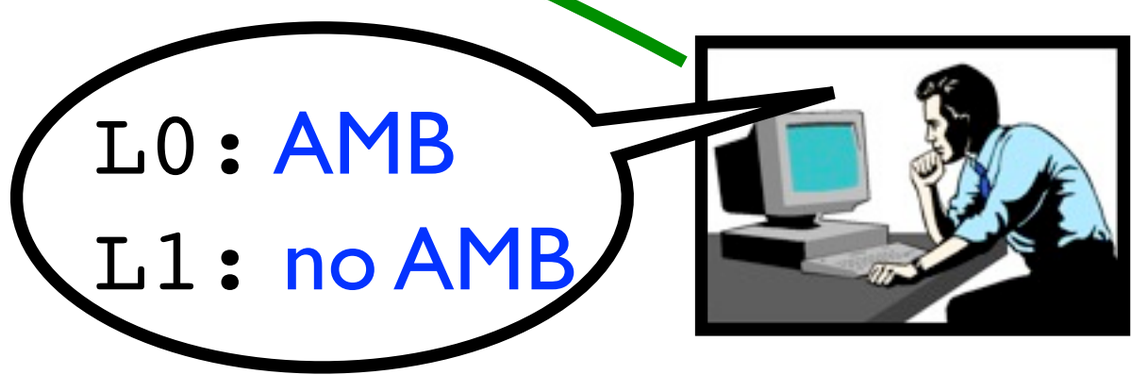
```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
    cap_enter();
L1: compress(in, out);
}
```



gzip
Programming Challenges

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
    cap_enter();
L1: compress(in, out);
}
```

no AMB



gzip
Programming Challenges

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
    cap_enter();
L1: compress(in, out);
}
```

no AMB

Capsicum Rules for Ambient Authority

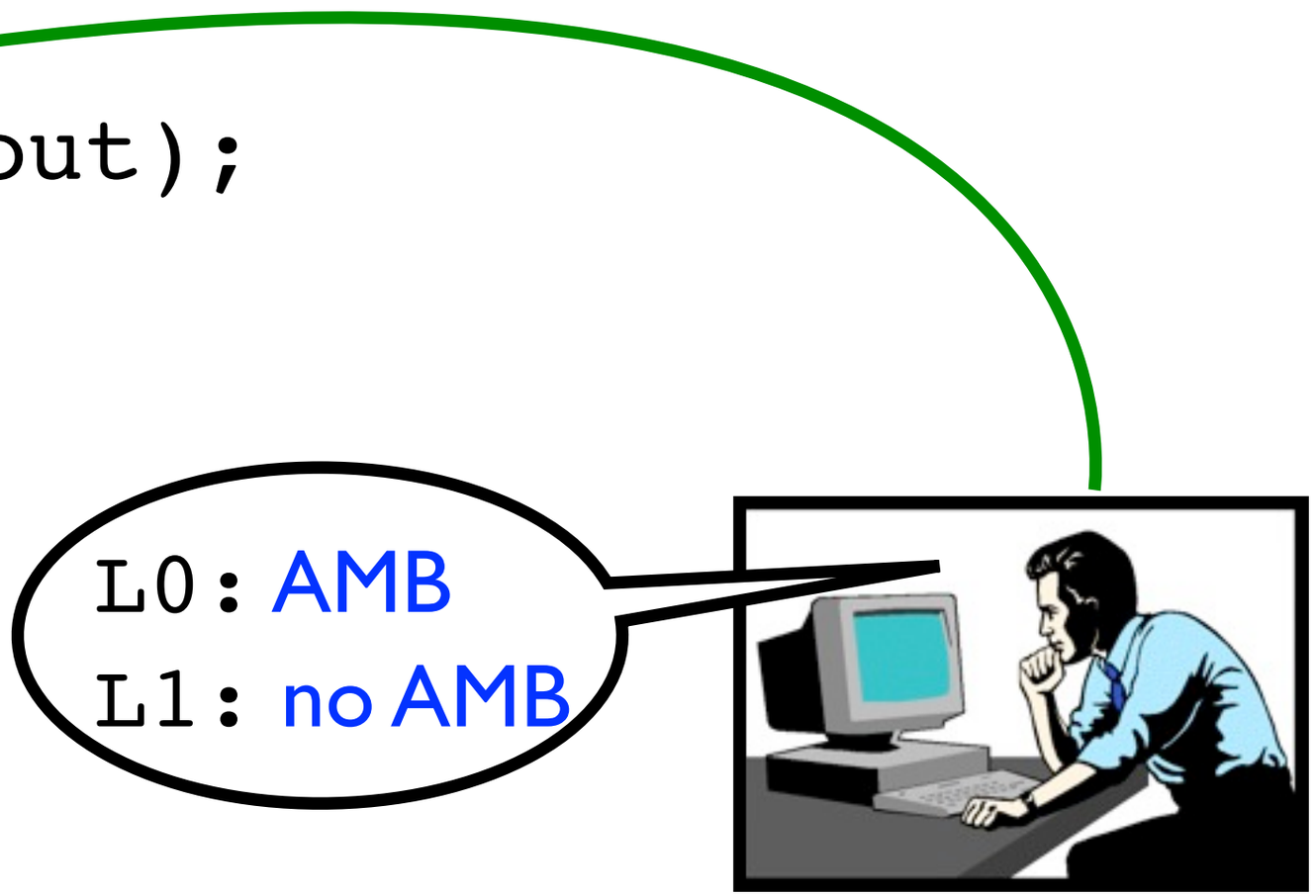
1. When a process is created,
   it has the AMB value of its parent
2. After a process calls `cap_enter()`,
   it never has AMB

Capsicum Rules for Ambient Authority

1. When a process is created,
   it has the AMB value of its parent

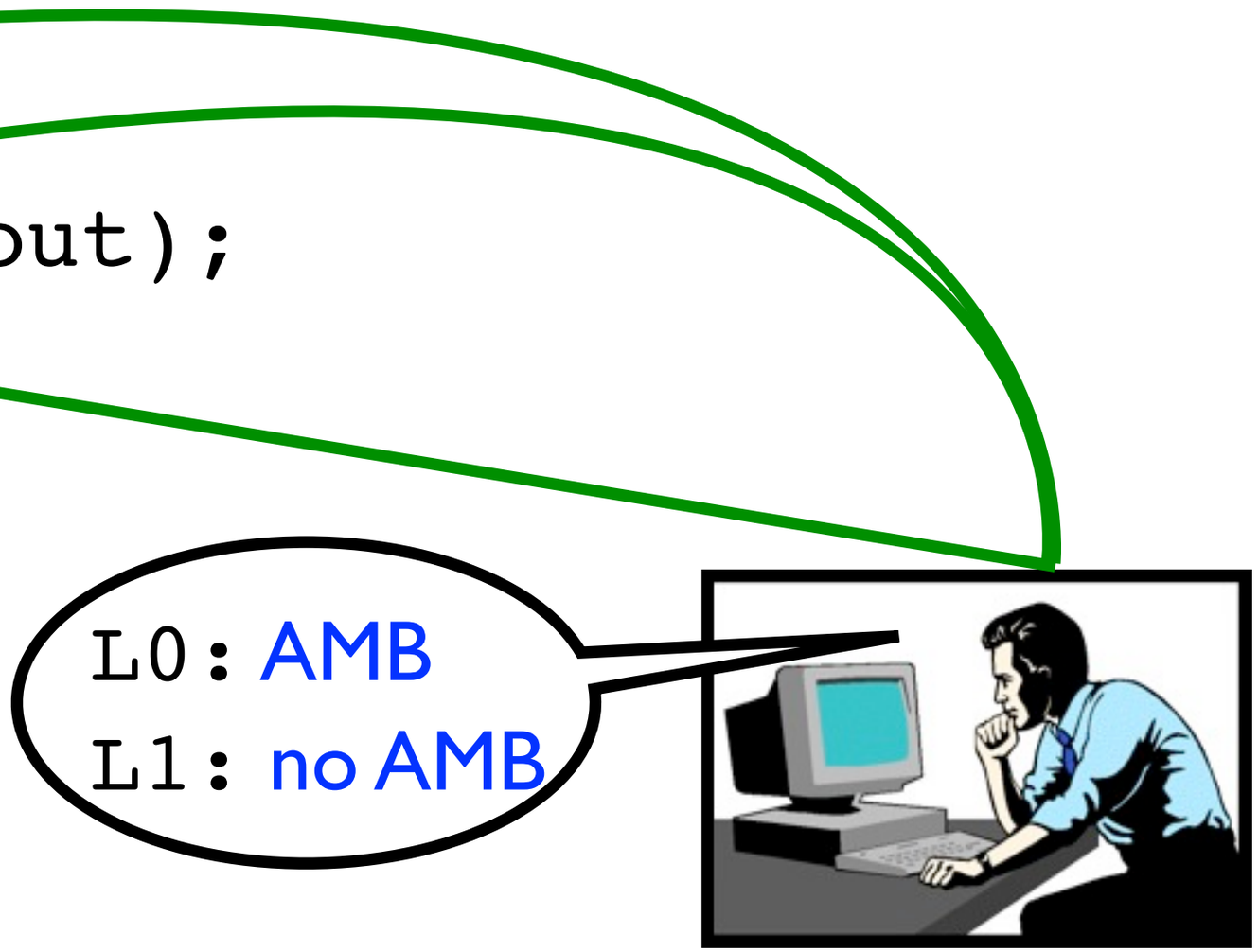
Instrumenting `gzip`

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);

      cap_enter();
L1: compress(in, out);

}
```



Instrumenting `gzip`

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
      sync_fork();
      cap_enter();
L1: compress(in, out);
      sync_join();
}
```



Instrumenting gzip

```
main() {
   file_nms = parse_cl(); AMB
   for (f in file_nms):
L0: (in, out) = open2(f);
      sync_fork();
      cap_enter();
L1: compress(in, out);
      sync_join();
}
```

Instrumenting `gzip`

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
      sync_fork();
      cap_enter();
L1: compress(in, out);
      sync_join();
}
```

AMB

Instrumenting `gzip`

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);AMB
      sync_fork();
      cap_enter();
L1: compress(in, out);
      sync_join();
}
```

Instrumenting gzip

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
      sync_fork();
      cap_enter();
L1: compress(in, out);
      sync_join();
}
```

no AMB

Instrumenting `gzip`

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
      sync_fork();
      cap_enter();
L1: compress(in, out);
      sync_join();
}
```

AMB

Instrumenting `gzip`

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);AMB
      sync_fork();
      cap_enter();
L1: compress(in, out);
      sync_join();
}
```

Challenges Not Appearing in this Talk

Challenges Not Appearing in this Talk

- Capsicum supports capabilities as descriptors with ~60 rights

Challenges Not Appearing in this Talk

- Capsicum supports capabilities as descriptors with ~60 rights
- Policies may be truly temporal

Challenges Not Appearing in this Talk

- Capsicum supports capabilities as descriptors with ~60 rights
- Policies may be truly temporal
- Instrumented program may need to maintain extra state

Challenges Not Appearing in this Talk

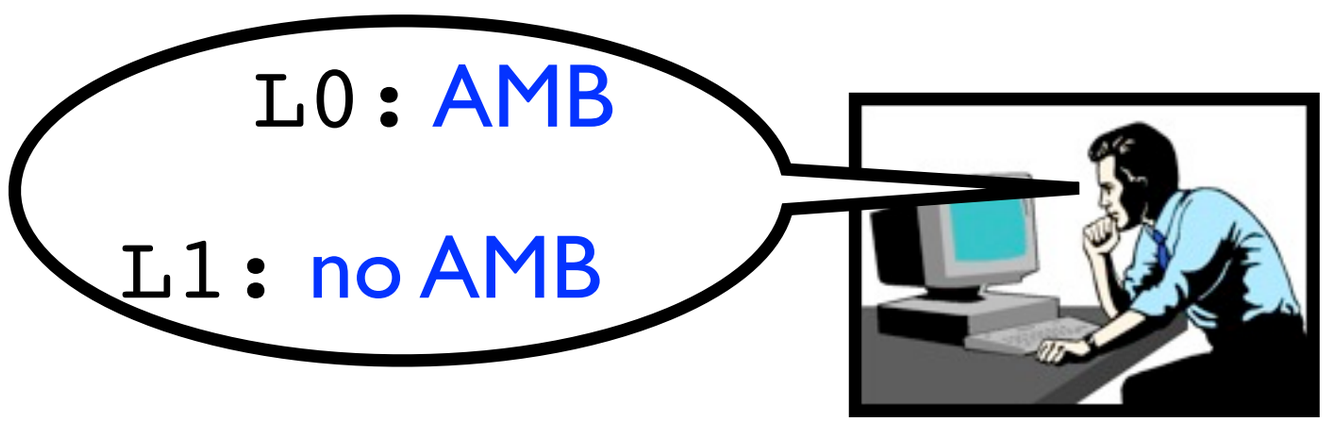
- Capsicum supports capabilities as descriptors with ~60 rights
- Policies may be truly temporal
- Instrumented program may need to maintain extra state
- Instrumented program may need to deal with injected code

Instrumenting Programs with CapWeave

1. Programmer writes an **explicit** policy
2. Compiler instruments program to invoke primitives so that it satisfies the policy

gzip with CapWeave

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```



gzip with CapWeave

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```

Policy

[L0: AMB]*

∩ [L1: no AMB]*

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```

Policy

[L0: AMB]*

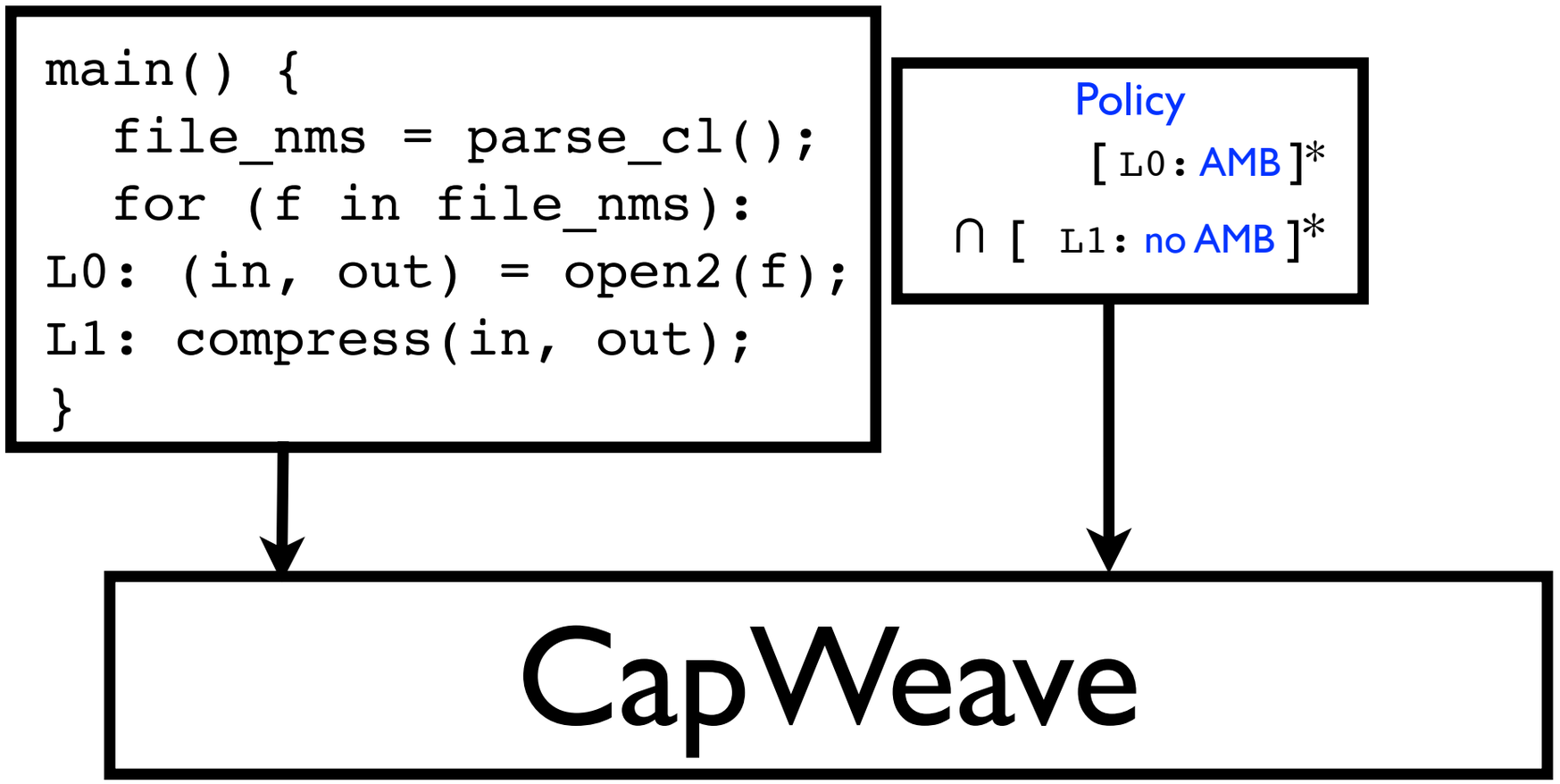
∩ [L1: no AMB]*

```
main() {
   file_nms = parse_cl();
   for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```

Policy

[L0: AMB]*

∩ [L1: no AMB]*


main() {
file_nms = parse_cl();
for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
Policy
[L0: AMB]*
∩ [L1: no AMB]*
CapWeave

```
main() {
  file_nms = parse_cl();
  for (f in file_nms):
L0: (in, out) = open2(f);
L1: compress(in, out);
}
```

Policy

[L0: AMB]*

∩ [L1: no AMB]*

CapWeave

Instrumented Program

```
void main() {
  L0: open2(...); (AMB)
  sync_fork();
  cap_enter();
  L1: compress(); (no AMB)
  sync_join();
}
```

Talk Outline

1. Why use Capsicum? (USENIX Security '10)
2. Why use CapWeave?
3. How does CapWeave work?
4. How well does CapWeave work?

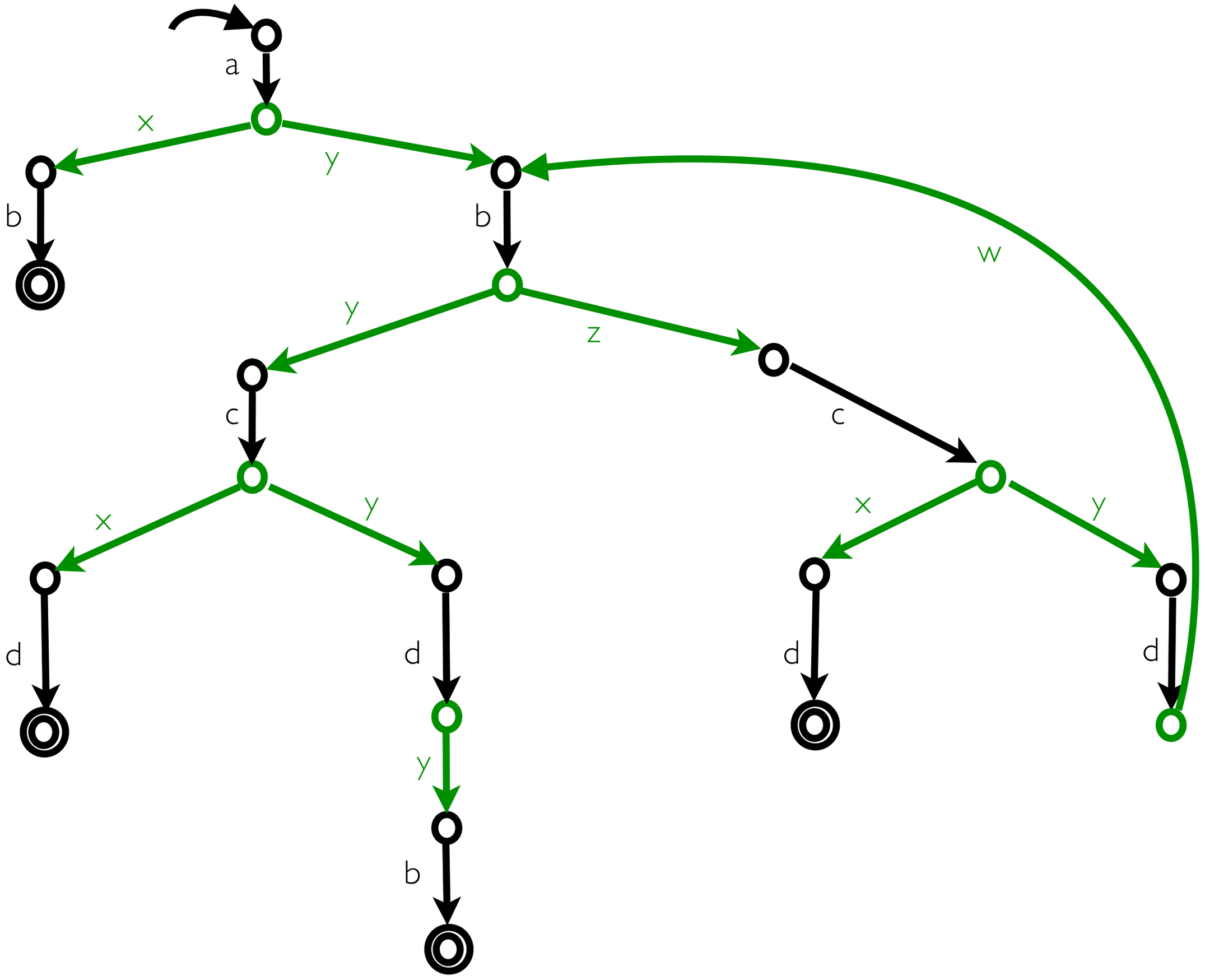
Talk Outline

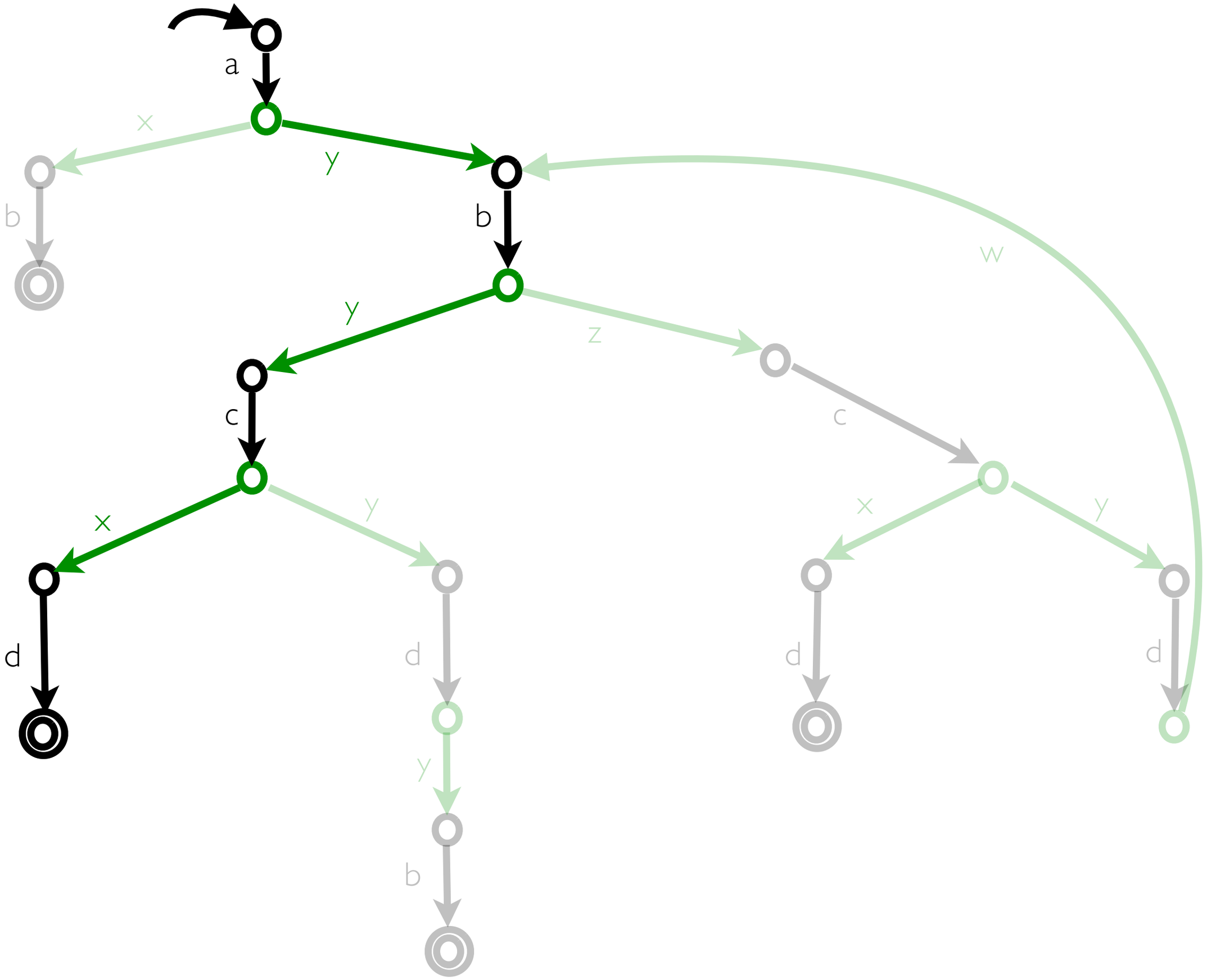
3. How does CapWeave work?

A. By reducing instrumentation to a game

Two-Player Safety Games

- In an Attacker state,
  the Attacker chooses the next input
- In a Defender state,
  the Defender chooses the next input
- Attacker wants to reach an accepting state

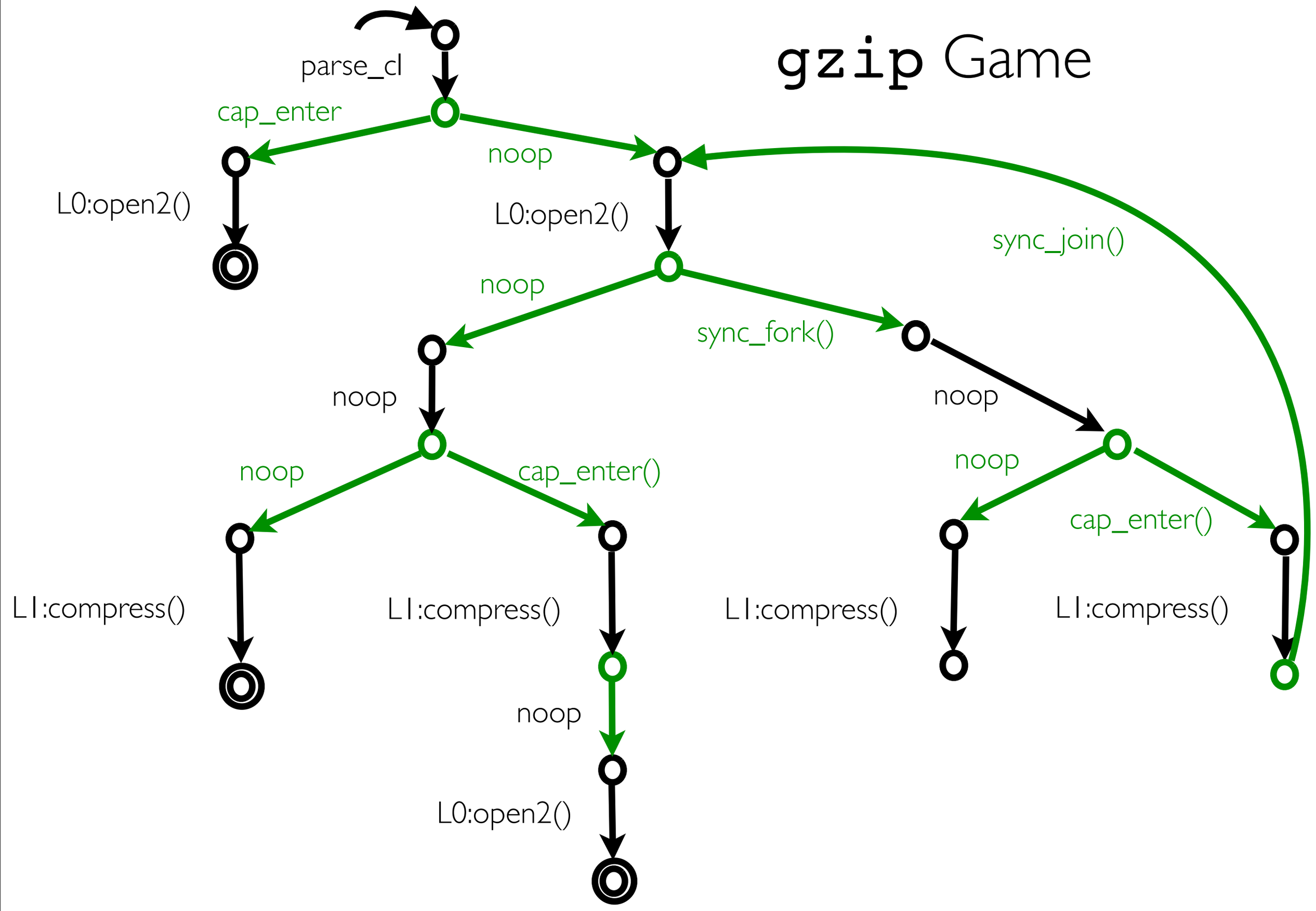

a
x
y
b
b
w
y
z
c
c
x
y
x
y
d
d
y
b
d
d


a
x
y
b
b
w
y
z
c
c
x
y
x
y
d
d
y
b
d
d

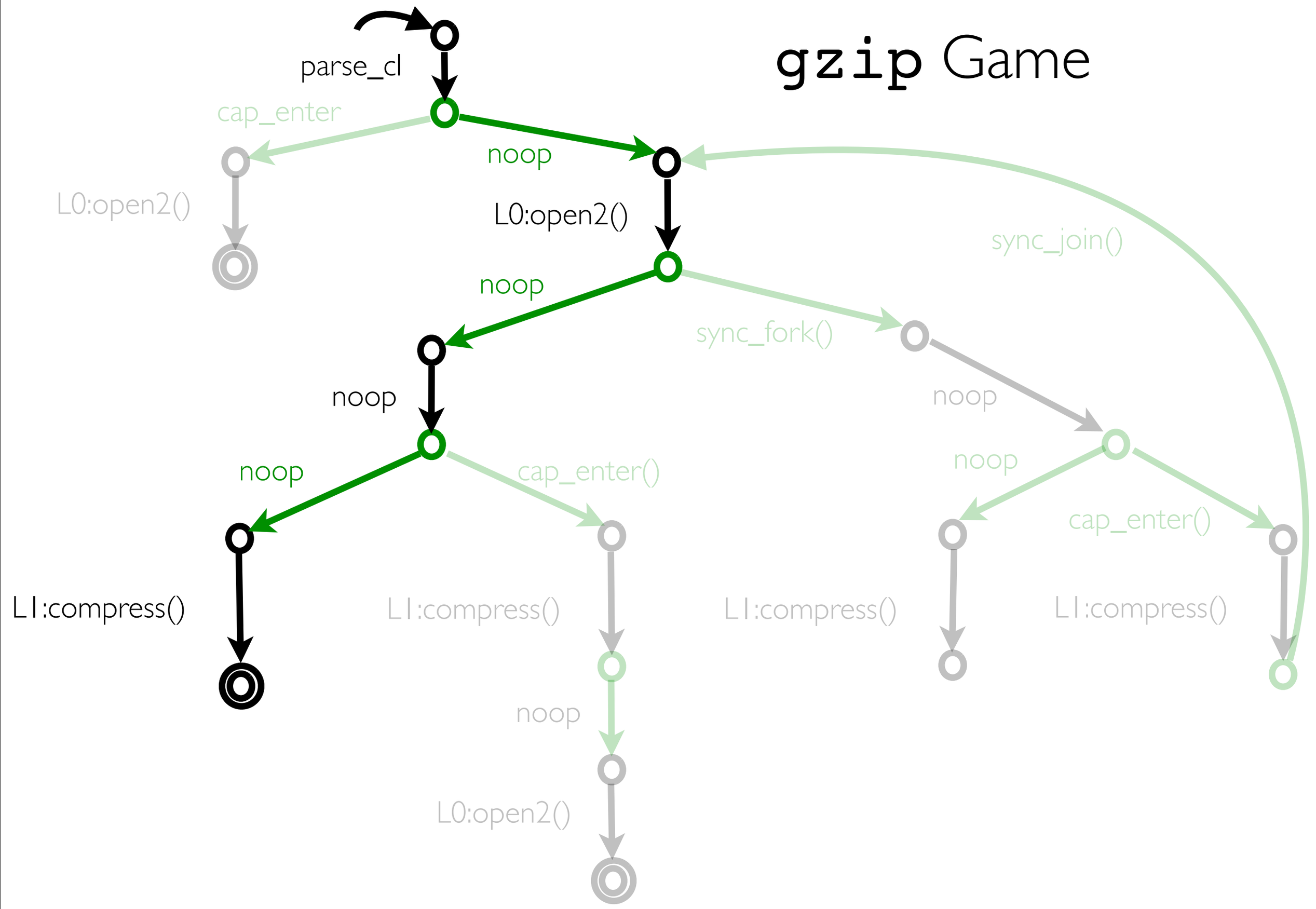
Instrumentation as a Game

Capsicum Instrumentation	**Two-player Games**
Program instructions	Attacker actions
Capsicum primitives	Defender actions
Policy violations	Attacker wins
Satisfying instrumentation	Winning Defender strategy

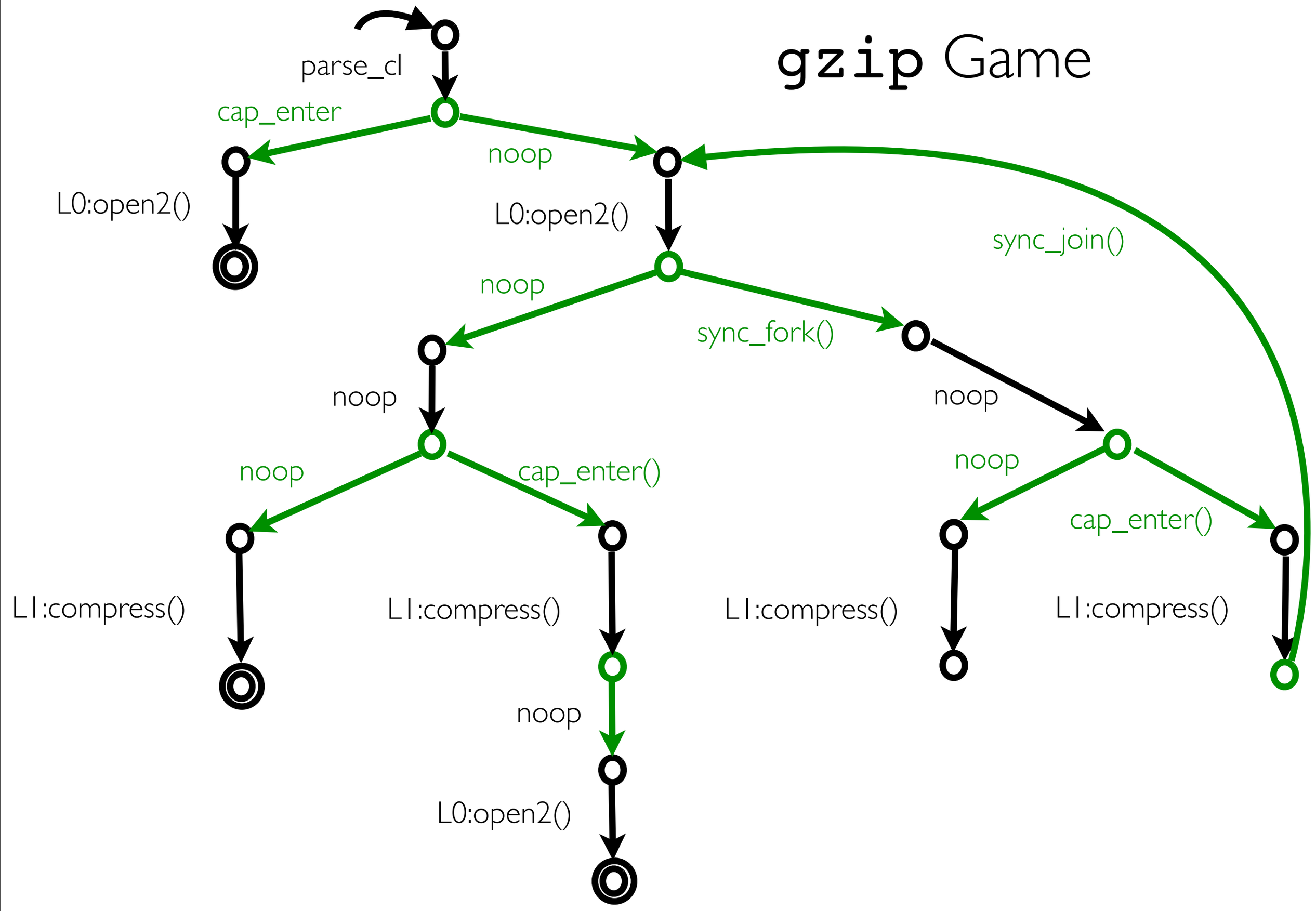
`gzip` Game


parse_cl
cap_enter
noop
L0:open2()
L0:open2()
sync_join()
noop
sync_fork()
noop
noop
noop
cap_enter()
noop
cap_enter()
L1:compress()
L1:compress()
L1:compress()
L1:compress()
noop
L0:open2()

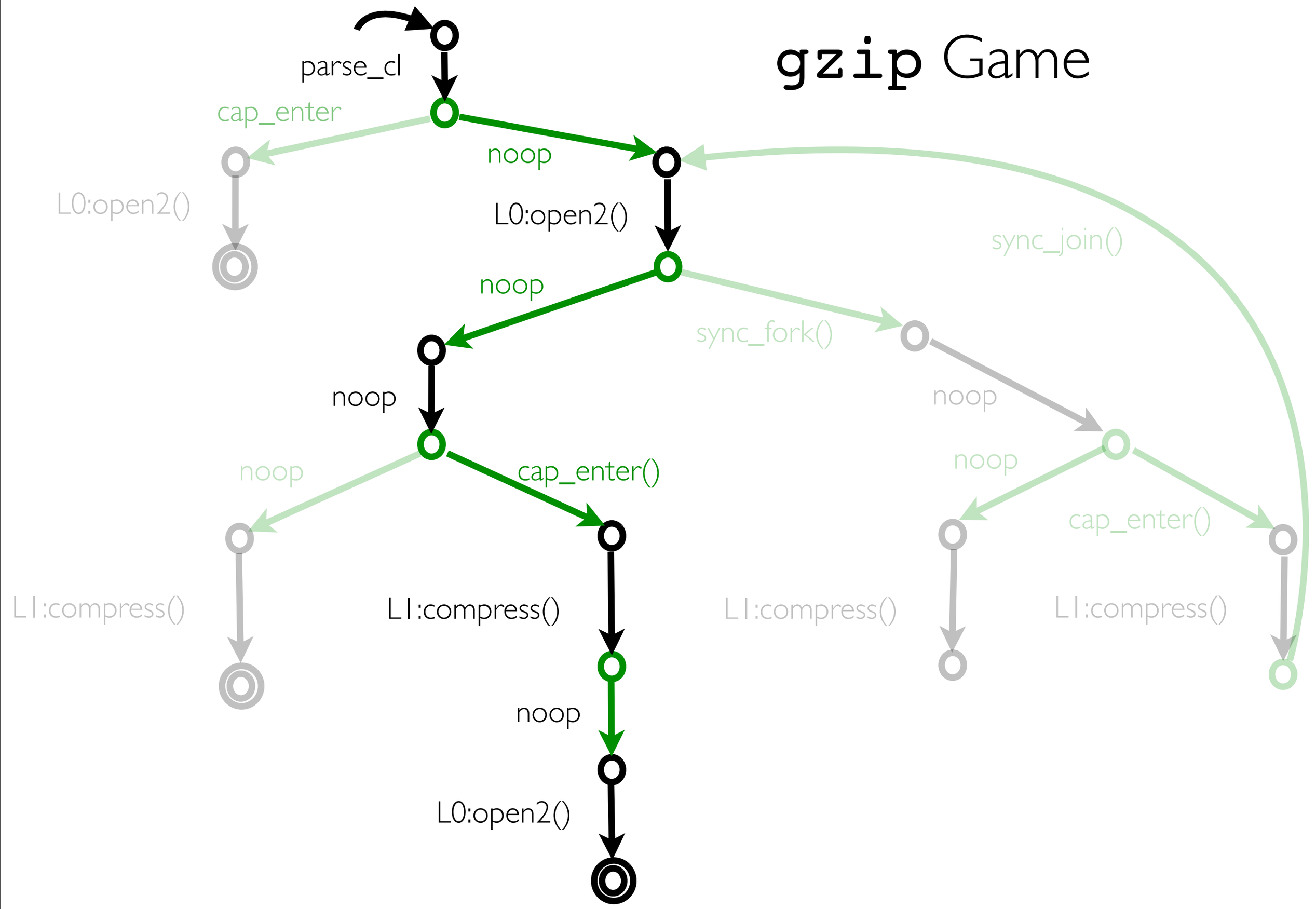
`gzip` Game


parse_cl
cap_enter
noop
L0:open2()
L0:open2()
sync_join()
noop
sync_fork()
noop
noop
noop
cap_enter()
cap_enter()
L1:compress()
L1:compress()
L1:compress()
L1:compress()
noop
L0:open2()

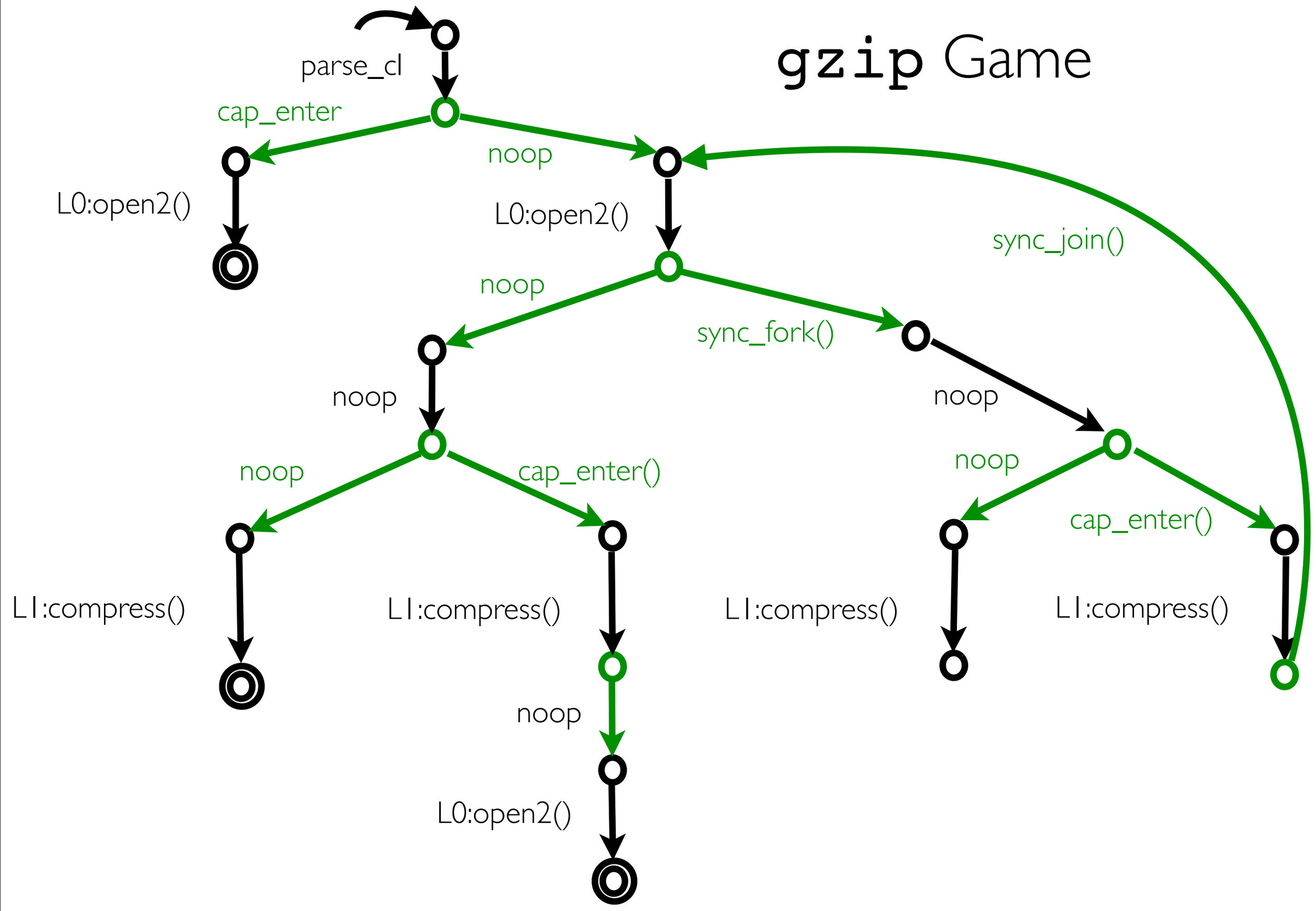
`gzip` Game


parse_cl
cap_enter
noop
L0:open2()
L0:open2()
sync_join()
noop
sync_fork()
noop
noop
noop
cap_enter()
noop
cap_enter()
L1:compress()
L1:compress()
L1:compress()
L1:compress()
noop
L0:open2()

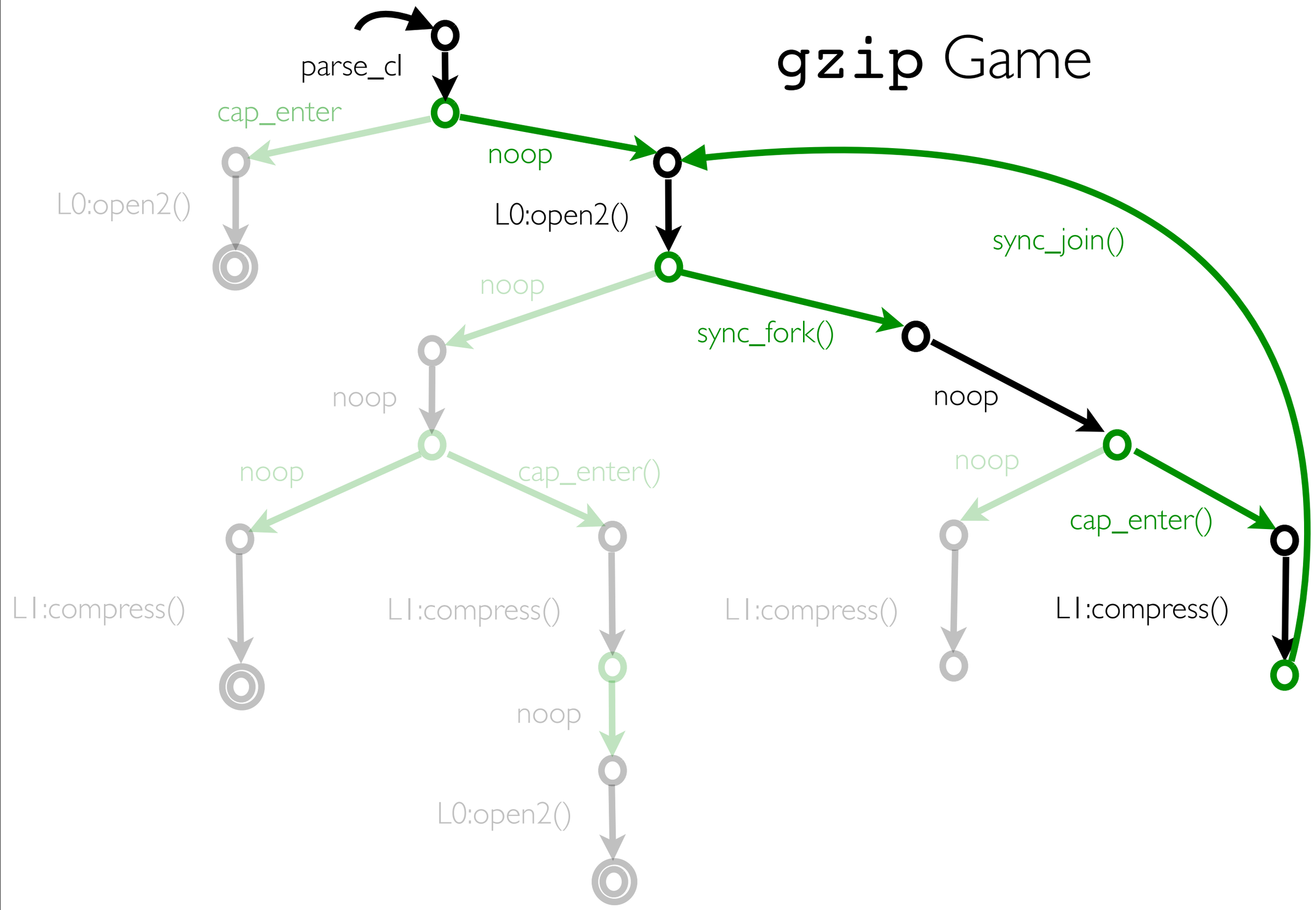
`gzip` Game


parse_cl
cap_enter
noop
L0:open2()
L0:open2()
sync_join()
noop
sync_fork()
noop
noop
noop
cap_enter()
cap_enter()
L1:compress()
L1:compress()
L1:compress()
L1:compress()
noop
L0:open2()

`gzip` Game


parse_cl
cap_enter
noop
L0:open2()
L0:open2()
sync_join()
noop
sync_fork()
noop
noop
noop
cap_enter()
noop
cap_enter()
L1:compress()
L1:compress()
L1:compress()
L1:compress()
noop
L0:open2()

`gzip` Game


parse_cl
cap_enter
noop
L0:open2()
L0:open2()
sync_join()
noop
sync_fork()
noop
noop
noop
cap_enter()
cap_enter()
L1:compress()
L1:compress()
L1:compress()
L1:compress()
noop
L0:open2()

Talk Outline

1. Why use Capsicum? (USENIX Security '10)
2. Why use CapWeave?
3. How does CapWeave work?
4. How well does CapWeave work?

Talk Outline

4. How well does CapWeave work?

Weaver Performance

Name	Program kLoC	Policy LoC	Weaving Time
`bzip2-1.0.6`	8	70	4m57s
`gzip-1.2.4`	9	68	3m26s
`php-cgi-5.3.2`	852	114	46m36s
`tar-1.25`	108	49	0m08s
`tcpdump-4.1.1`	87	52	0m09s
`wget-1.12`	64	35	0m10s

Performance on Included Tests

Name	Base Time	Hand Overhd	capweave Overhd	Diff. Overhd (%)
bzip2-1.0.6	0.593s	0.909	1.099	20.90
gzip-1.2.4	0.036s	1.111	1.278	15.03
php-cgi-5.3.2	0.289s	1.170	1.938	65.64
tar-1.25	0.156s	13.301	21.917	64.78
tcpdump-4.1.1	1.328s	0.981	1.224	24.77
wget-1.12	4.539s	1.906	1.106	0.91

Performance on Practical Workloads

- Ran woven `bzip2`, `gzip`, and `wget` on 1GB of Capsicum source code
- Overhead for each was ≤ 4% over baseline

Current Limitations

- Optimal placement of primitives
- Diagnosing inconsistent policies

Program

```
void main(...) {
L0: open2(...);
L1: compress(...);
}
```

```
[ L0:AMB ]*
∩ [ L1: no AMB ]*
```

Policy

CapWeave

Instrumented Program

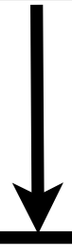
```
void main() {
  L0: open2(...); (AMB)
  sync_fork();
  cap_enter();
  L1: compress(); (no AMB)
  sync_join();
}
```

Program

```
void main(...) {
L0: open2(...);
L1: compress(...);
}
```

[L0: AMB]*
∩ [L1: no AMB]*

Policy



CapWeave

Program

```
void main(...) {
L0: open2(...);
L1: compress(...);
}
```

[L0:AMB]*
∩ [**L0**: no AMB]*

Policy

CapWeave

Program

```
void main(...) {
L0: open2(...);
L1: compress(...);
}
```

[L0:AMB]*
∩ [**L0**: no AMB]*

Policy

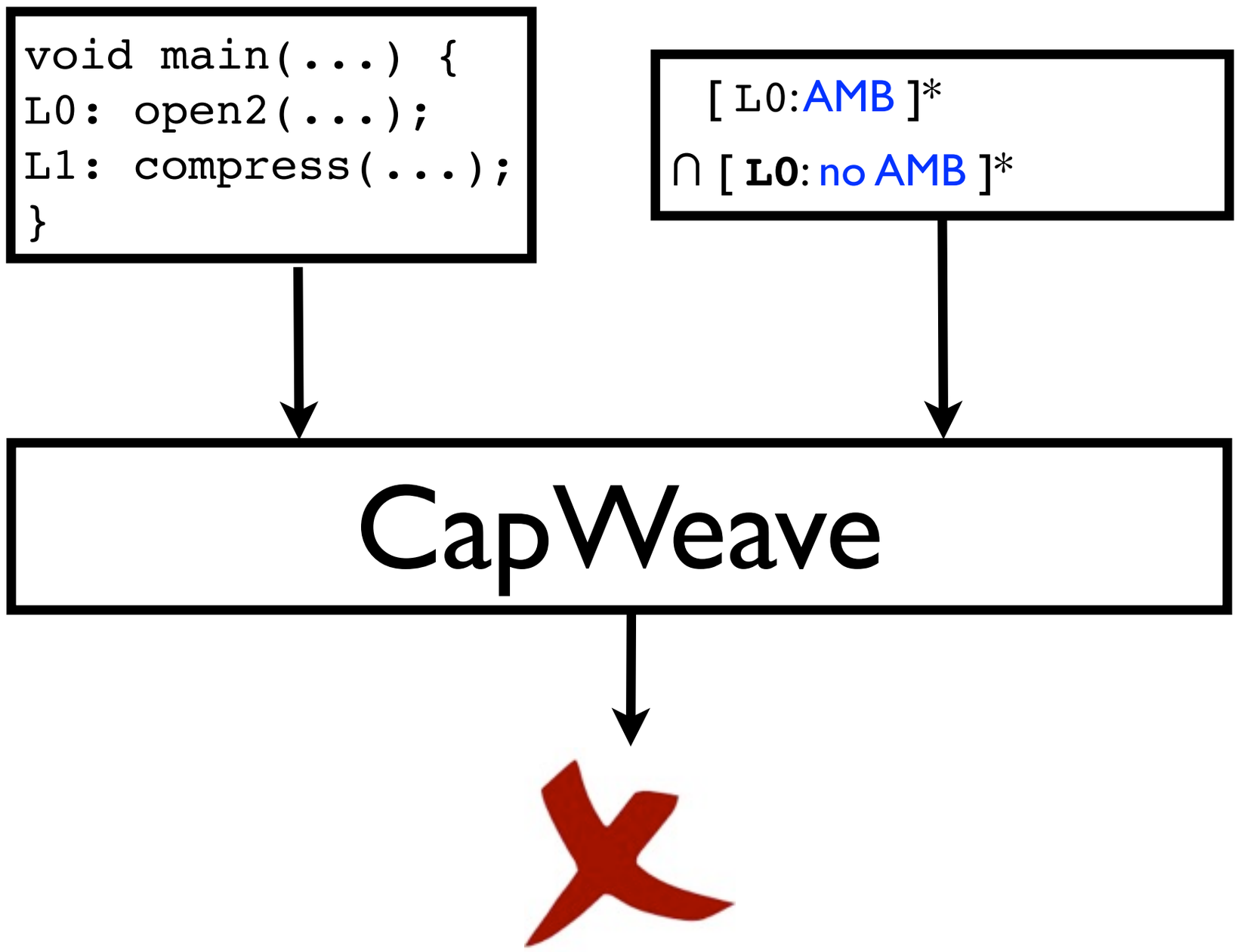
CapWeave

Program

```
void main(...) {
L0: open2(...);
L1: compress(...);
}
```

[L0:AMB]*
∩ [**L0**: no AMB]*

Policy



Talk Outline

1. Why use Capsicum? (USENIX '10)
2. Why use CapWeave?
3. How does CapWeave work?
4. How well does CapWeave work?

A big thanks to:

Capsicum-dev

Pawel Jakub Dawidek

Khilan Gudka

Ben Laurie

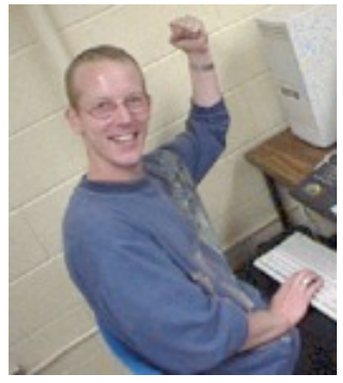
Peter Neumann

MIT-LL

Jeffrey Seibert

Michael Zhivich

Our shepherd



Niels Provos

Questions?

Program

```
main() {
L0: open2(...);
L1: compress(...);
}
```

```
[ L0:AMB ]*
∩ [ L1: AMB ]*
```

Policy

CapWeave

Instrumented Program

```
void main() {
  L0: open2(...); (AMB)
  sync_fork();
  cap_enter();
  L1: compress(...); (no AMB)
  sync_join();
}
```

Extra Slides

```
L0: for (int i = 0; i < num_urls; i++) {
        int svr_sock = open_http(urls[i]);
        char* out_path = urls[i];
        if (must_3xx_redirect(svr_sock)) {
 L1:      out_path = get_outnm(svr_sock); }
        read_http(svr_sock);
L2:    write_data(out_path);
}
```

```
for (int i = 0; i < num_urls; i++) {
    fork();
    int svr_sock = open_http(urls[i]);
    char* out_path = urls[i];
    bool is_redir = FALSE;
    if (must_3xx_redirect(svr_sock)) {
       is_redir = TRUE;
       out_path = get_outnm(svr_sock); }
    read_http(svr_sock);
    is_redir ? cap_enter : ;
    write_data(urls[i]);
    join(); }
}
```

```
L0: for (int i = 0; i < num_urls; i++) {
        fork();
        int svr_sock = open_http(urls[i]);
        char* out_path = urls[i];
        bool is_redir = FALSE;
        if (must_3xx_redirect(svr_sock)) {
           is_redir = TRUE;
L1:        out_path = get_outnm(svr_sock); }
        read_http(svr_sock);
L2:     write_data(out_path);
        join();
}
```

A Capsicum policy for wget

- When wget calls read_http(), it should be have AMB
- When wget calls write_data(), it should have AMB iff it never received a redirect request

A Capsicum policy for wget

- When wget calls read_http(), it should be have AMB
- When wget calls write_data(), it should have AMB iff it never received a redirect request

A CapWeave policy for wget

- When wget calls read_http(), it should be have AMB
- When wget calls write_data(), it should have AMB iff it never received a redirect request

A CapWeave policy for wget

.* [`L0` without AMB]

- When wget calls write_data(), it should have AMB iff it never received a redirect request

A CapWeave policy for wget

.* [L0 without AMB]

| .* [L1] [not L0]* [L2 with AMB]

| .* [L0] [not L1] [L2 without AMB]