Software Dataplane Verification

Mihai Dobrescu
Katerina Argyraki

EPFL

Software dataplanes

intrusion detection

application acceleration

IP forwarding

Software dataplanes

intrusion detection

application acceleration

IP forwarding

Software dataplanes

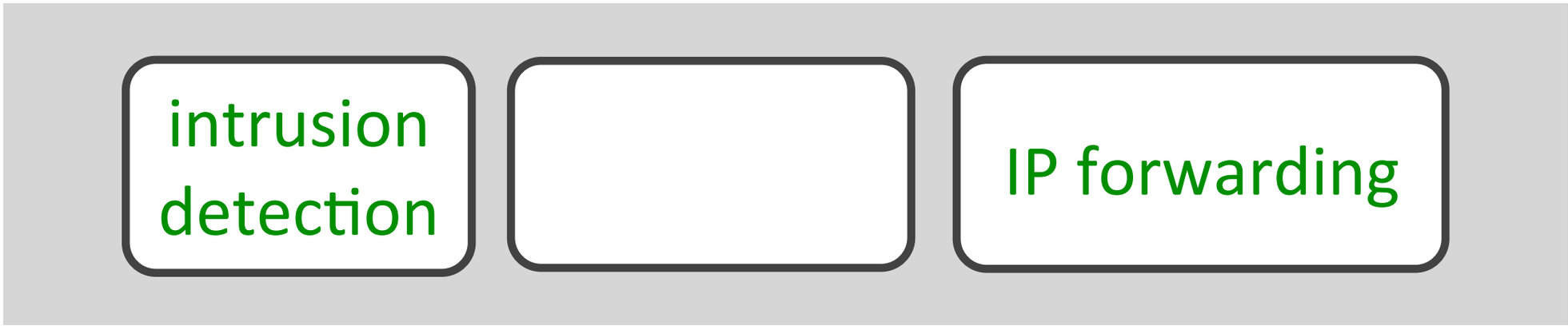
intrusion detection

application acceleration

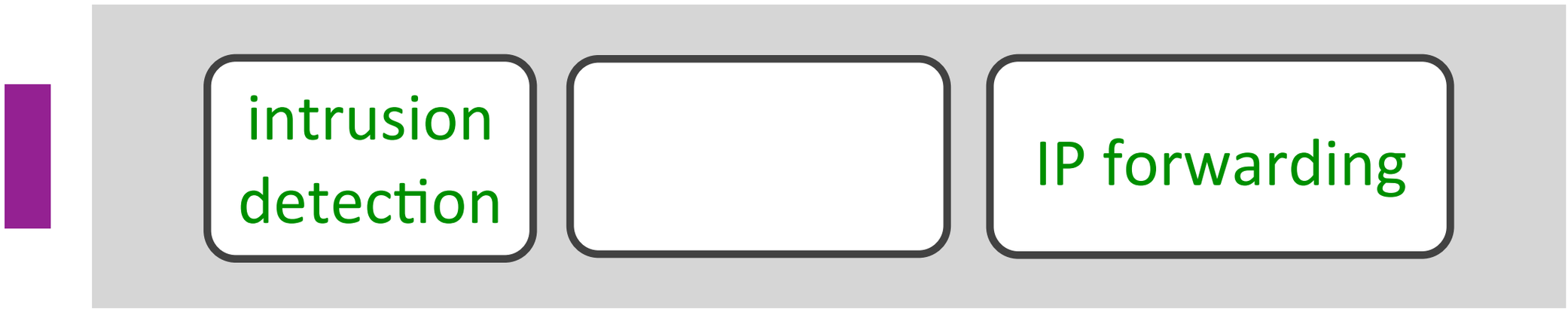
IP forwarding

Software dataplanes

Software dataplanes

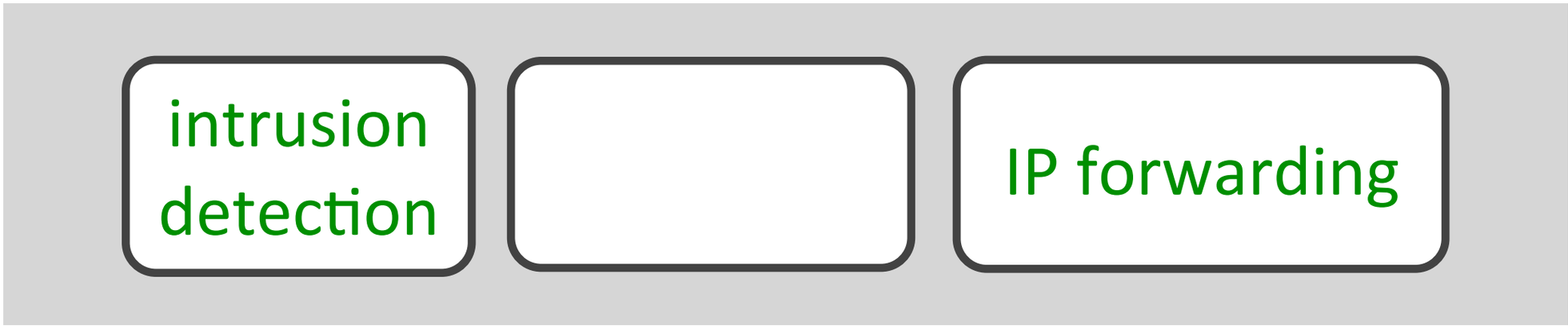


Software dataplanes



Software dataplanes

Software dataplanes



Software dataplanes

- Flexibility
  - *new intrusion detection, traffic filtering, sampling, application acceleration, ...*

- Unpredictability
  - *special packet causes router to crash*
  - *or doubles per-packet latency*

Dataplane verification

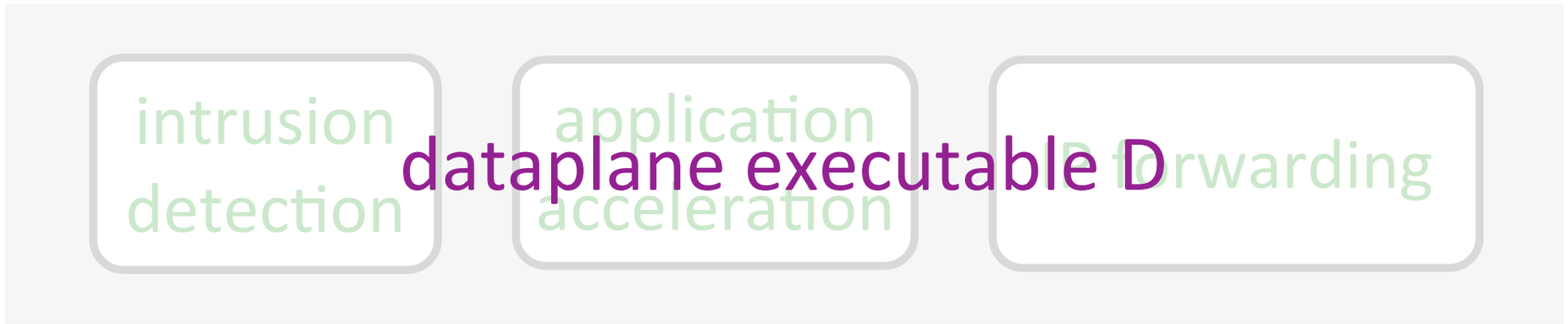
intrusion detection

application acceleration

IP forwarding

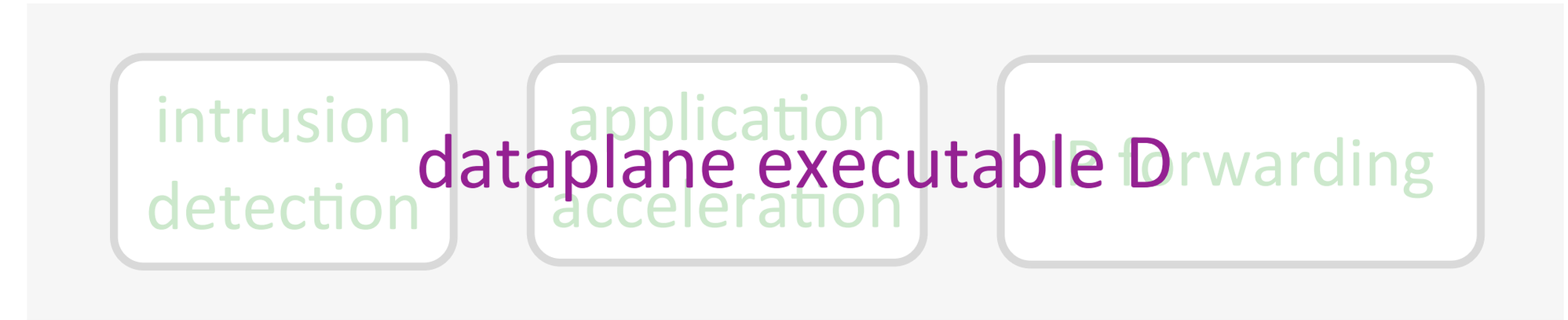
Dataplane verification

Dataplane verification

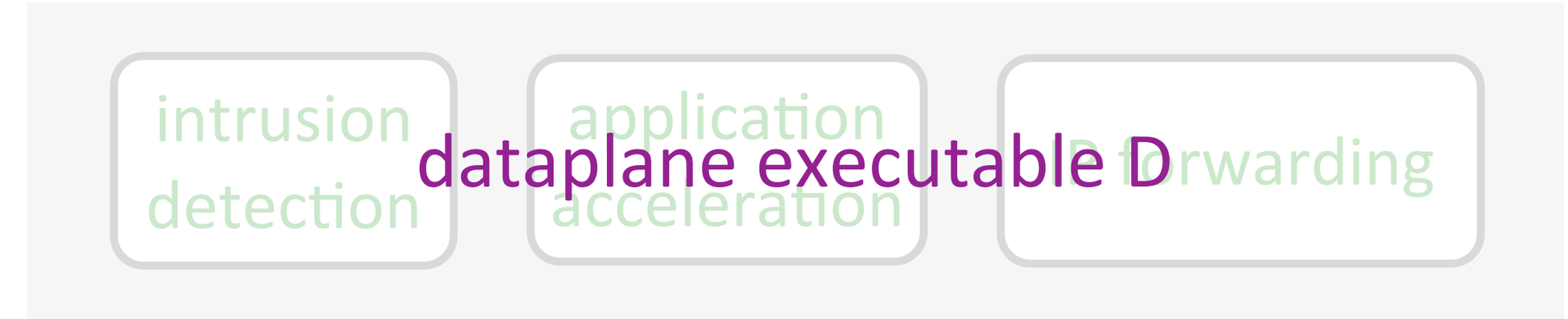


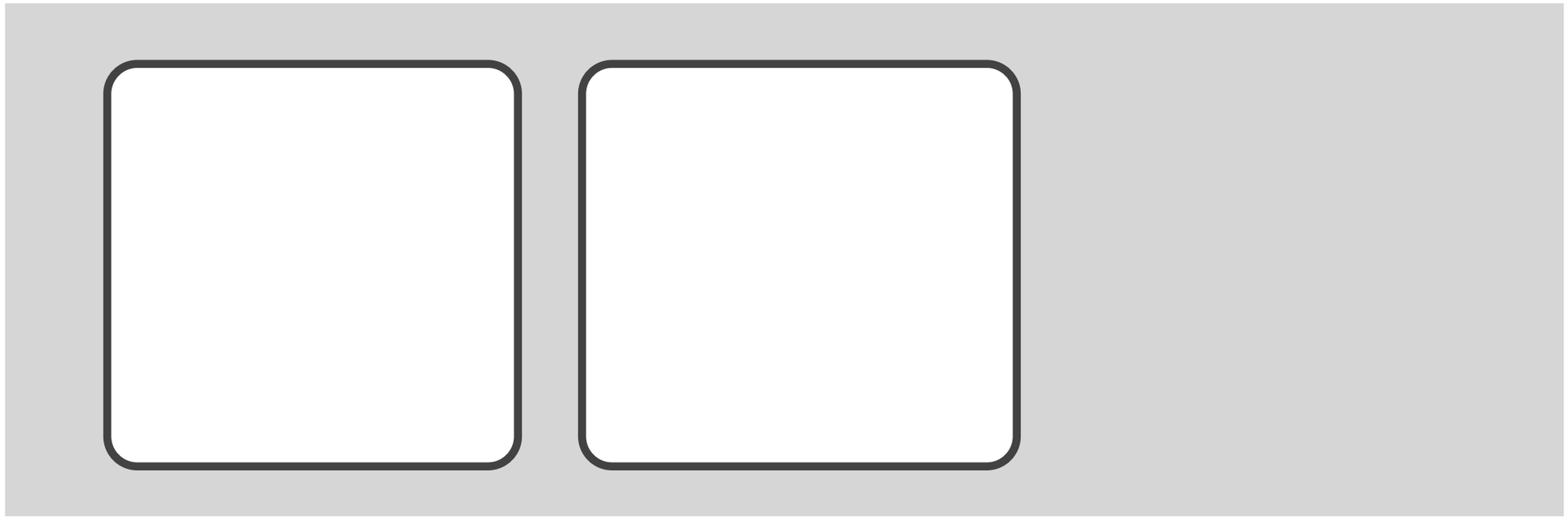
verification tool

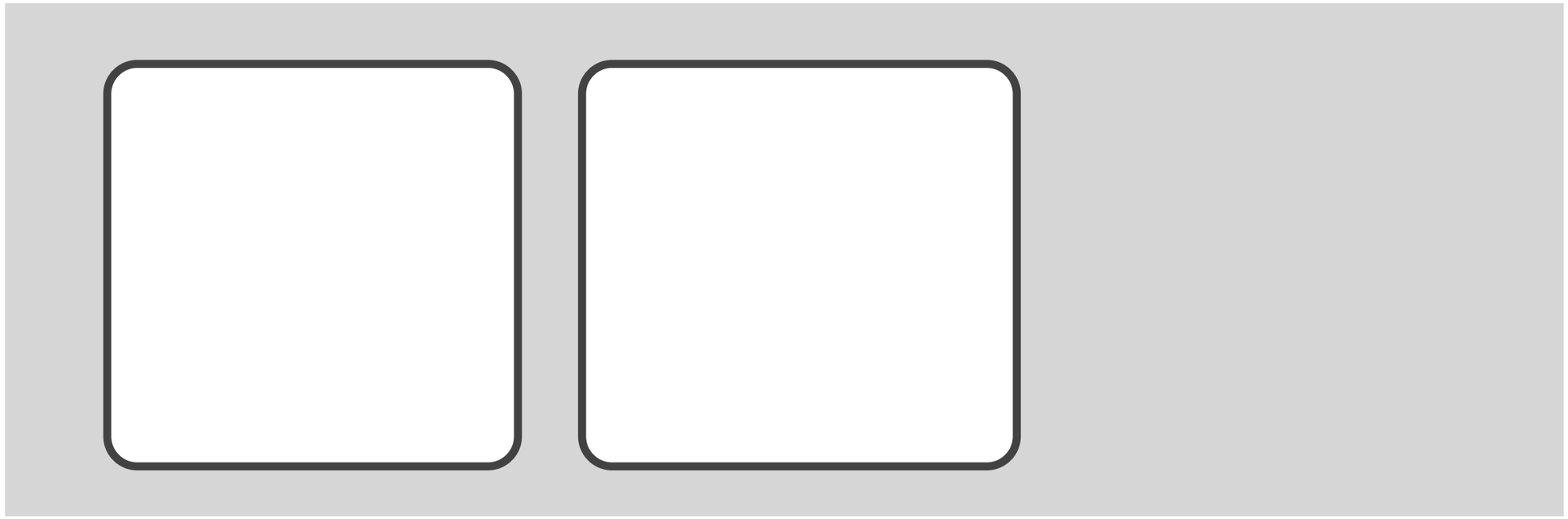
Dataplane verification



Dataplane verification







DART, PLDI 2005

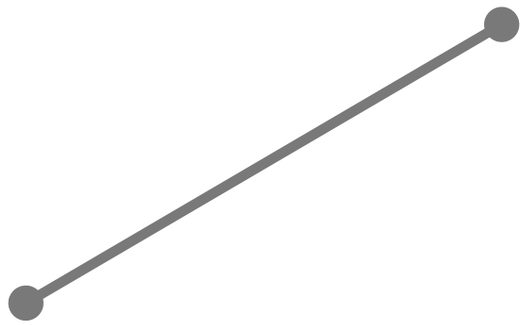
Klee, OSDI 2008

```
if (in.x < 0)
    out = ...;
else
    out = in;
```

DART, PLDI 2005

Klee, OSDI 2008

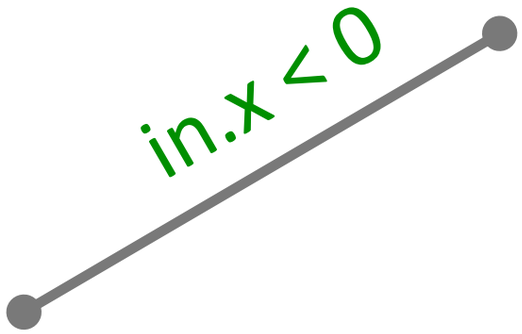
```
if (in.x < 0)
    out = ...;
else
    out = in;
```



DART, PLDI 2005

Klee, OSDI 2008

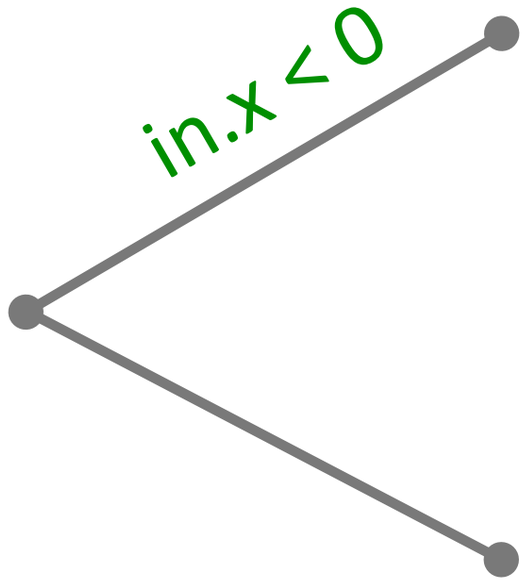
```
if (in.x < 0)
    out = ...;
else
    out = in;
```



DART, PLDI 2005

Klee, OSDI 2008

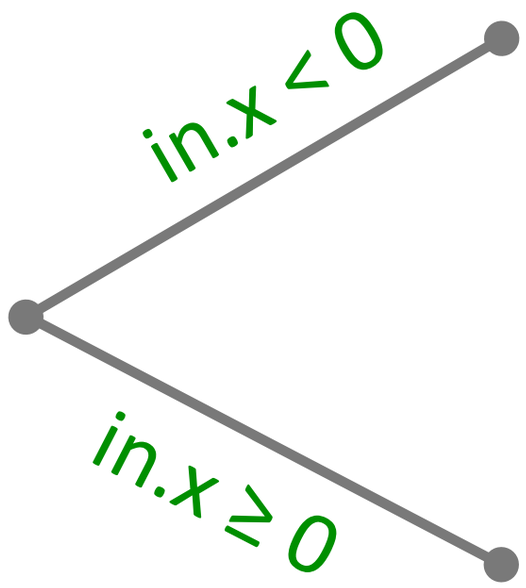
```
if (in.x < 0)
    out = ...;
else
    out = in;
```



DART, PLDI 2005

Klee, OSDI 2008

```
if (in.x < 0)
    out = ...;
else
    out = in;
```

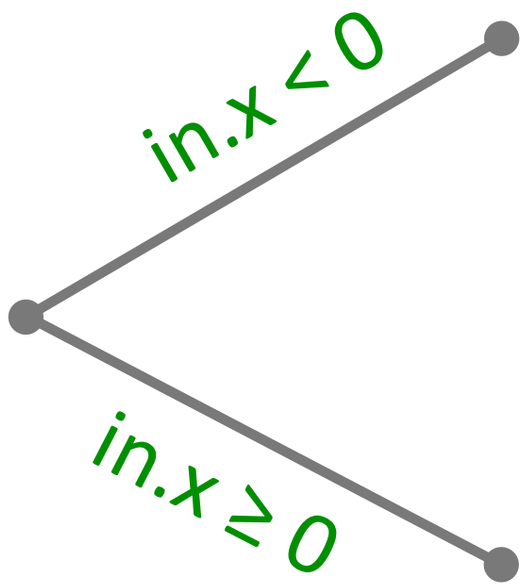


DART, PLDI 2005

Klee, OSDI 2008

```
if (in.x < 0)
    out = ...;
else
    out = in;
```

```
if (in.y < 10)
    out = ...;
else
    out = in;
```

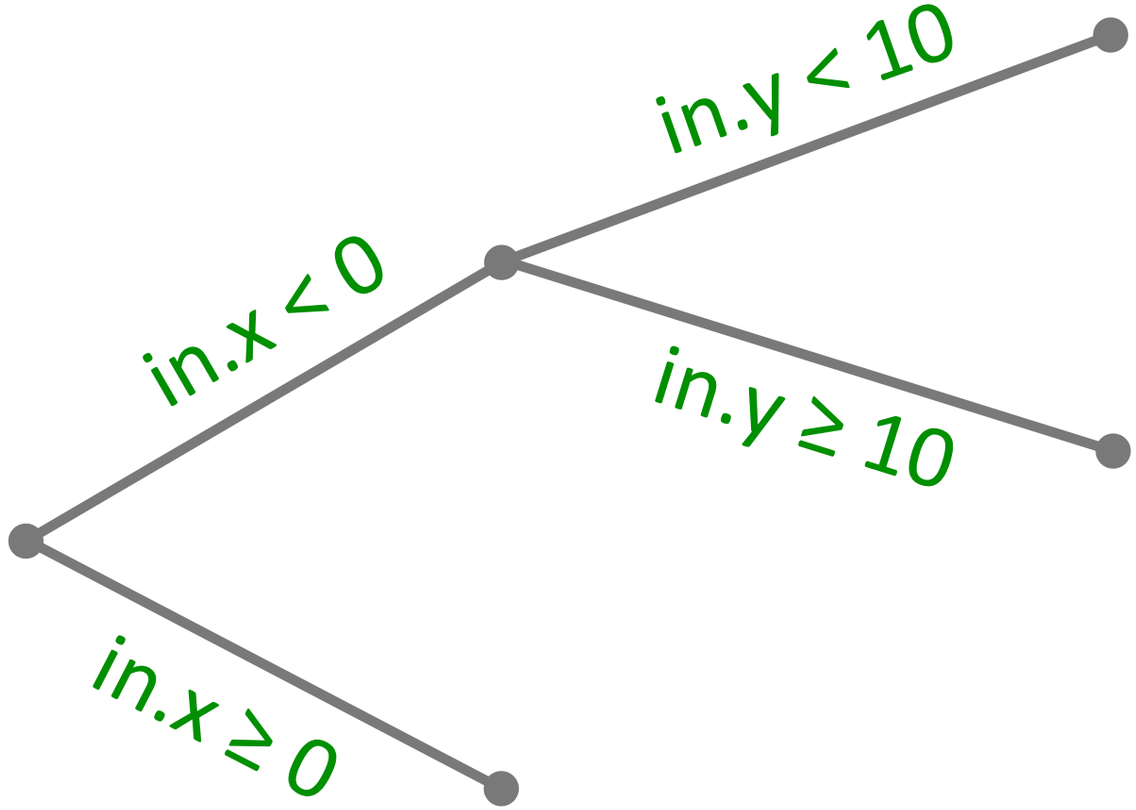


DART, PLDI 2005

Klee, OSDI 2008

```
if (in.x < 0)
    out = ...;
else
    out = in;
```

```
if (in.y < 10)
    out = ...;
else
    out = in;
```

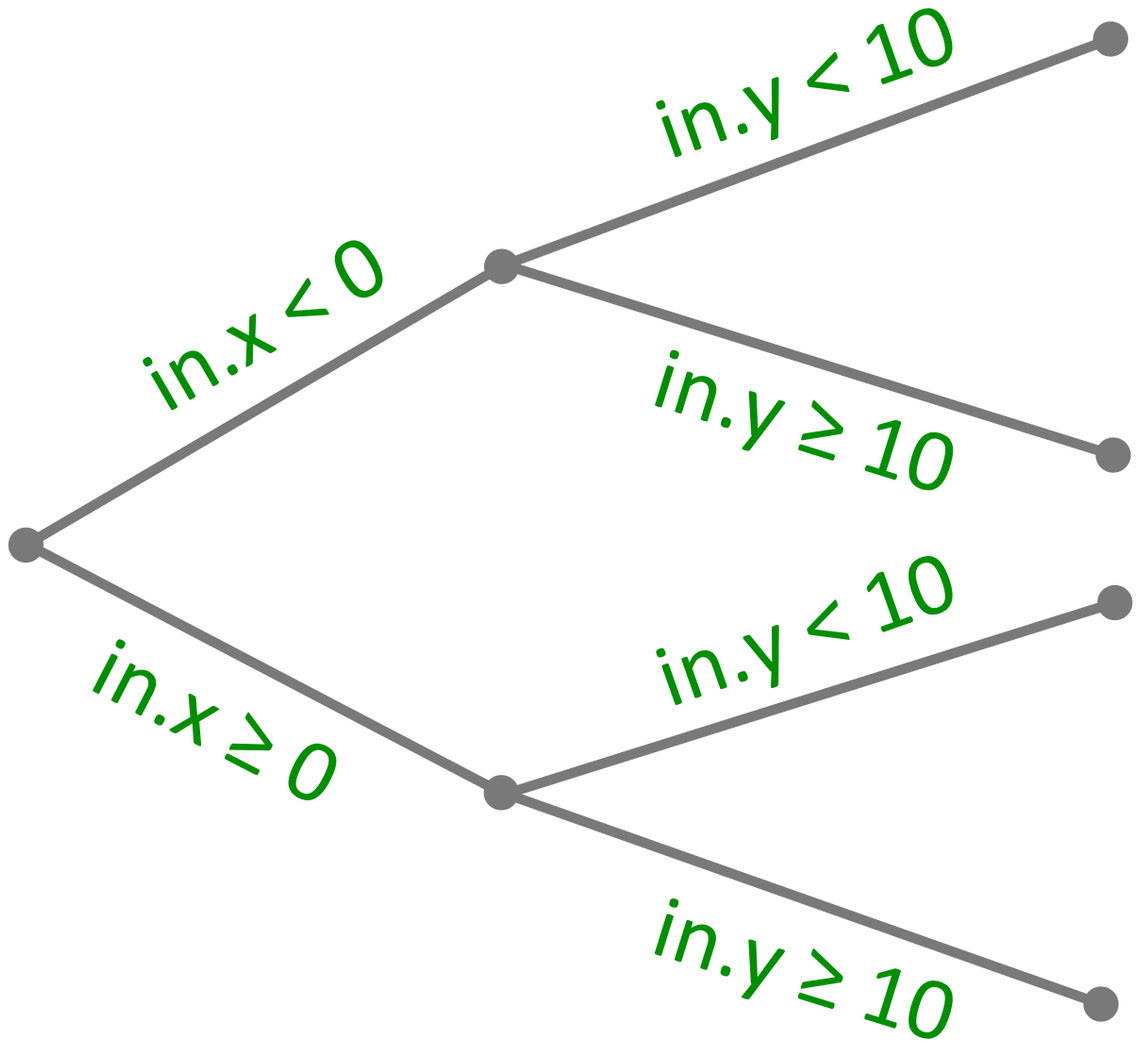


DART, PLDI 2005

Klee, OSDI 2008

```
if (in.x < 0)
    out = ...;
else
    out = in;
```

```
if (in.y < 10)
    out = ...;
else
    out = in;
```

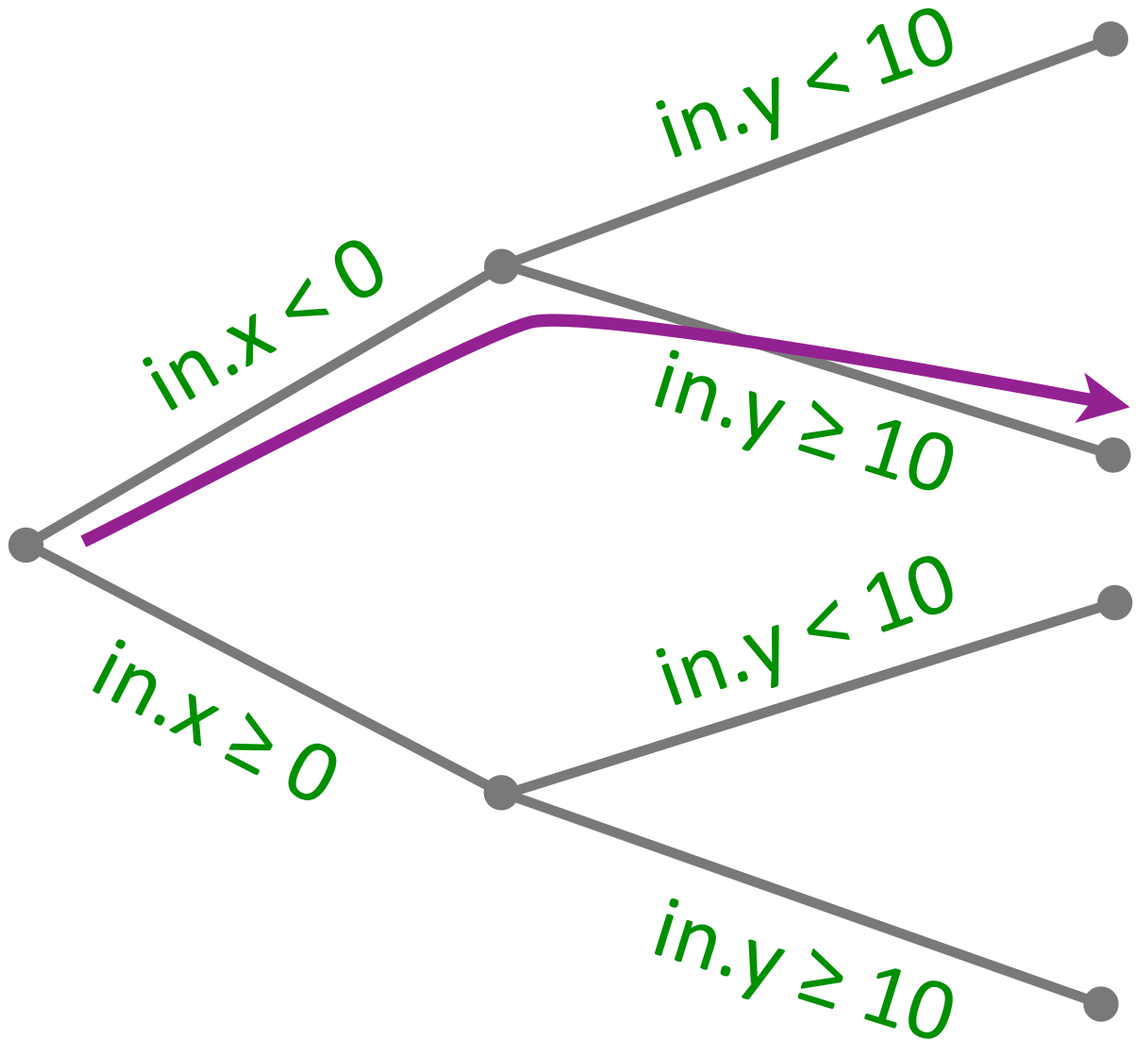


DART, PLDI 2005

Klee, OSDI 2008

```
if (in.x < 0)
    out = ...;
else
    out = in;
```

```
if (in.y < 10)
    out = ...;
else
    out = in;
```

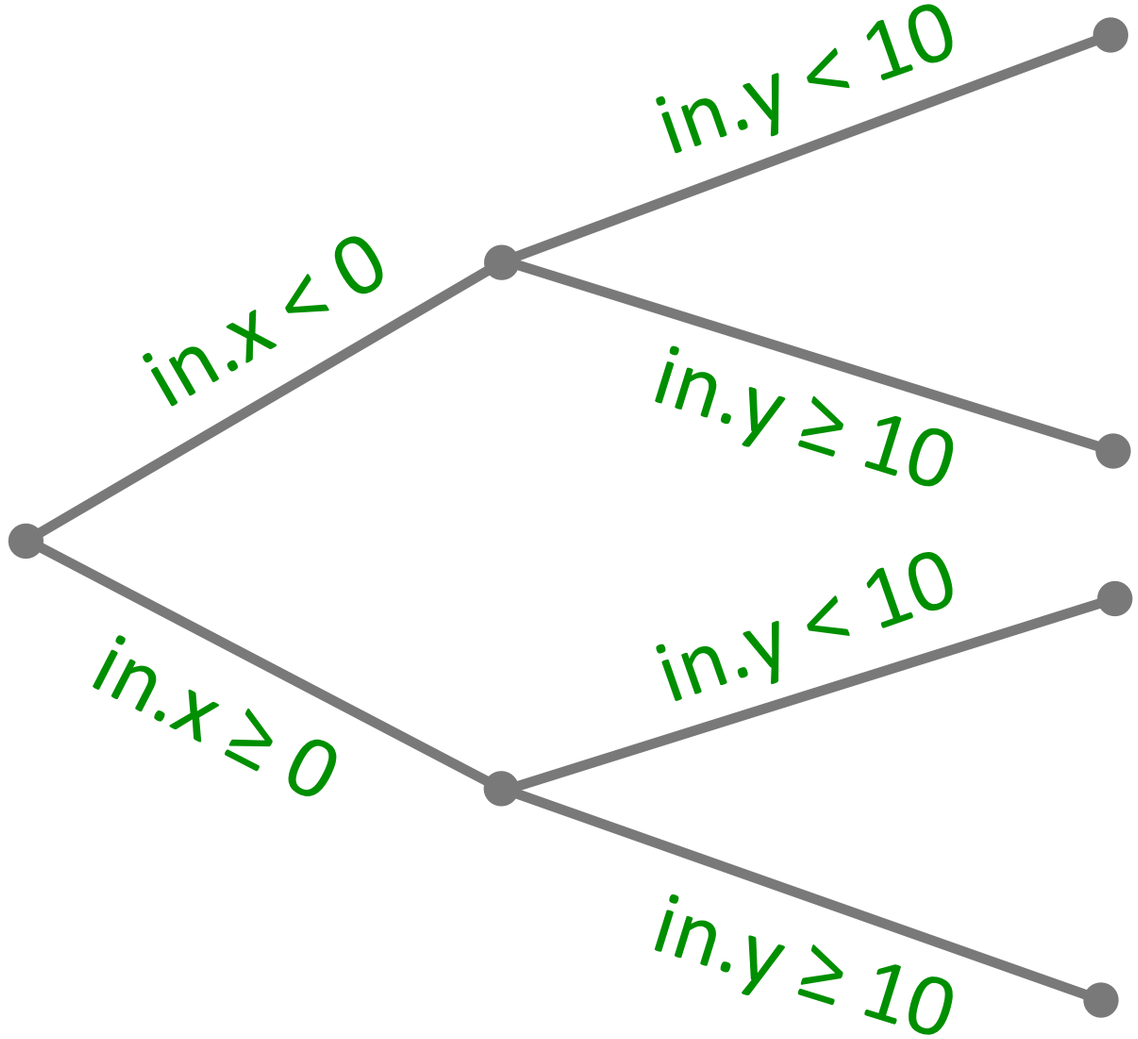


DART, PLDI 2005

Klee, OSDI 2008

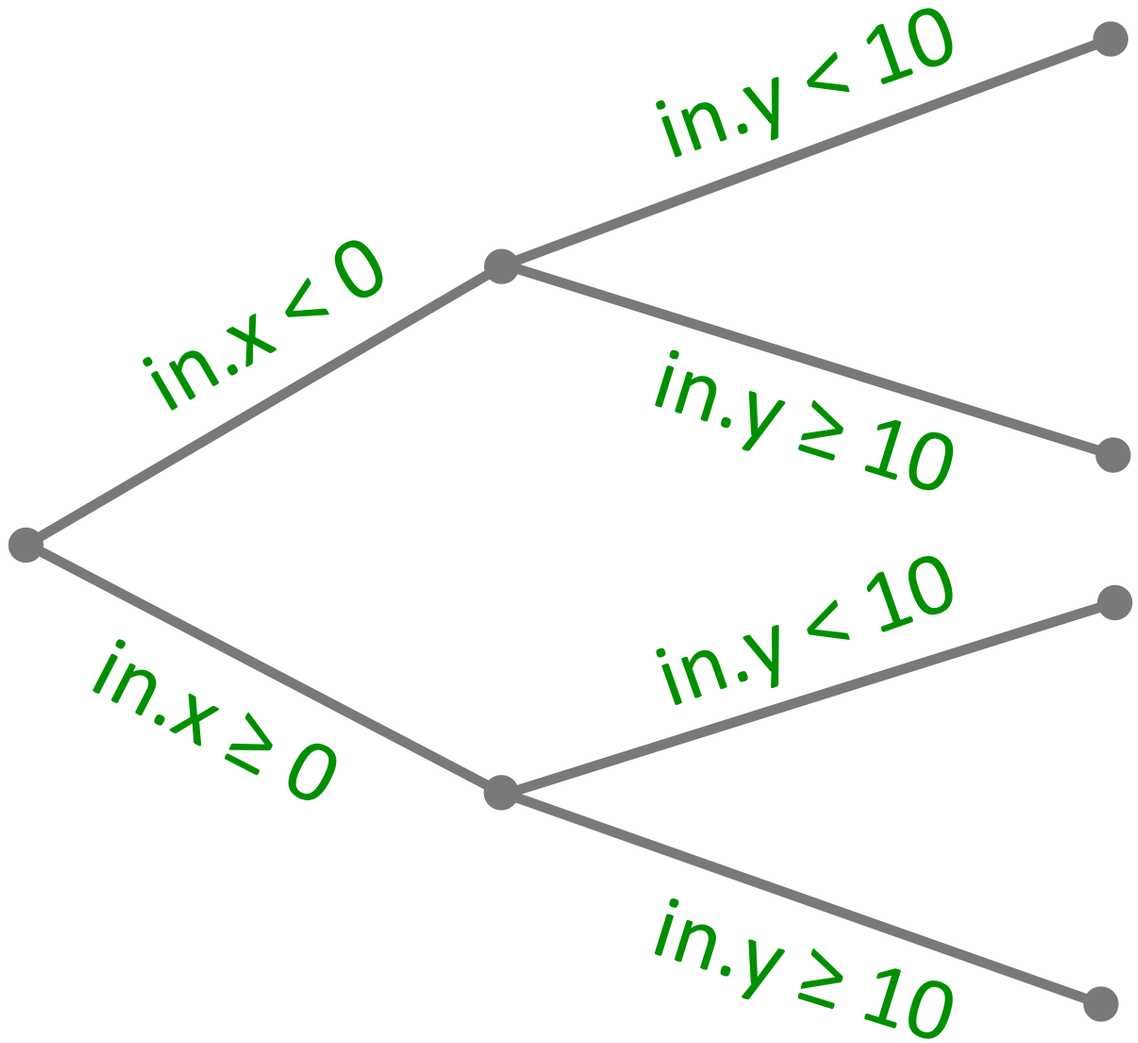
```
if (in.x < 0)
    out = ...;
else
    out = in;
```

```
if (in.y < 10)
    out = ...;
else
    out = in;
```



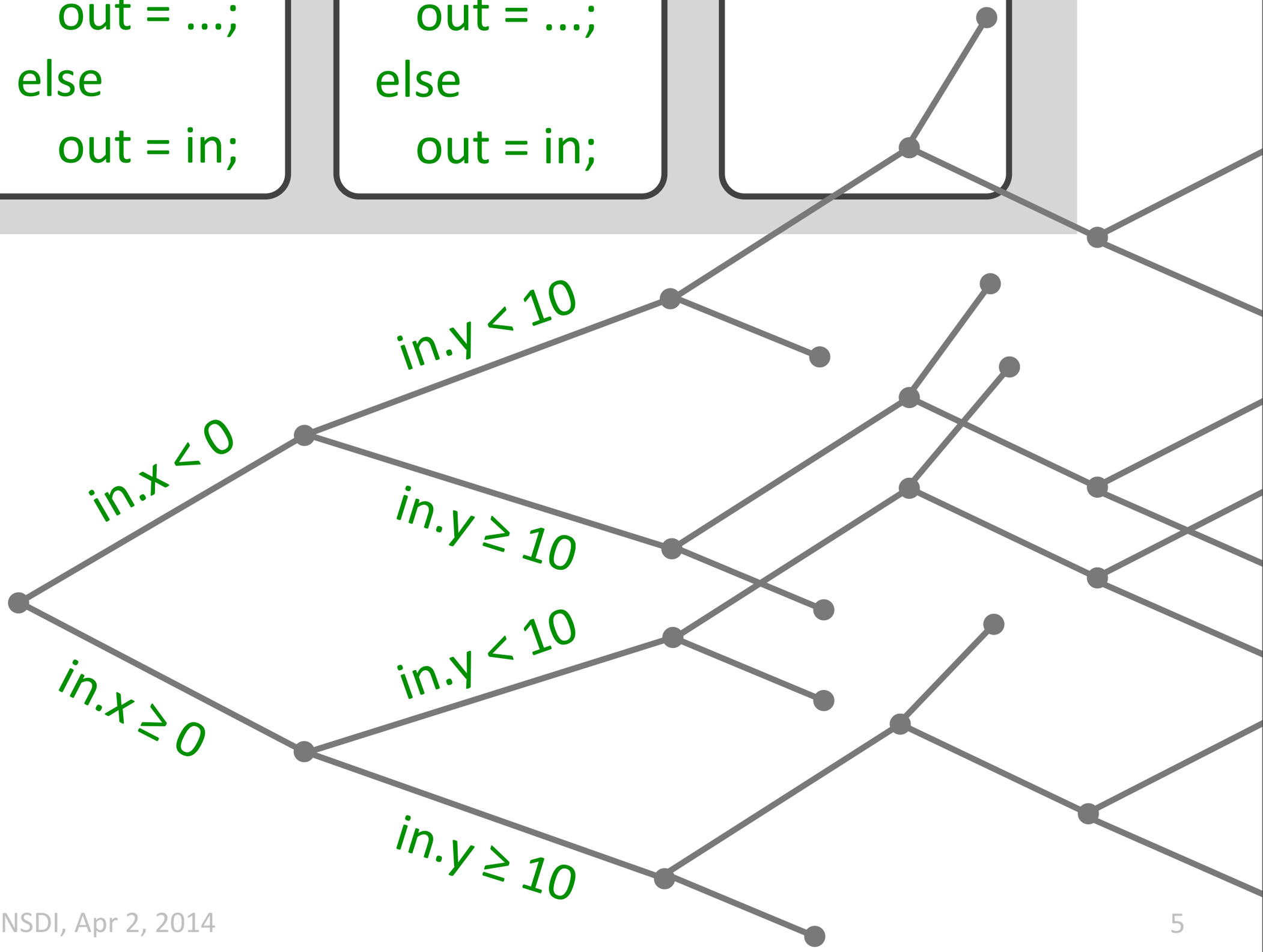
```
if (in.x < 0)
    out = ...;
else
    out = in;
```

```
if (in.y < 10)
    out = ...;
else
    out = in;
```



```
if (in.x < 0)
   out = ...;
else
   out = in;
```

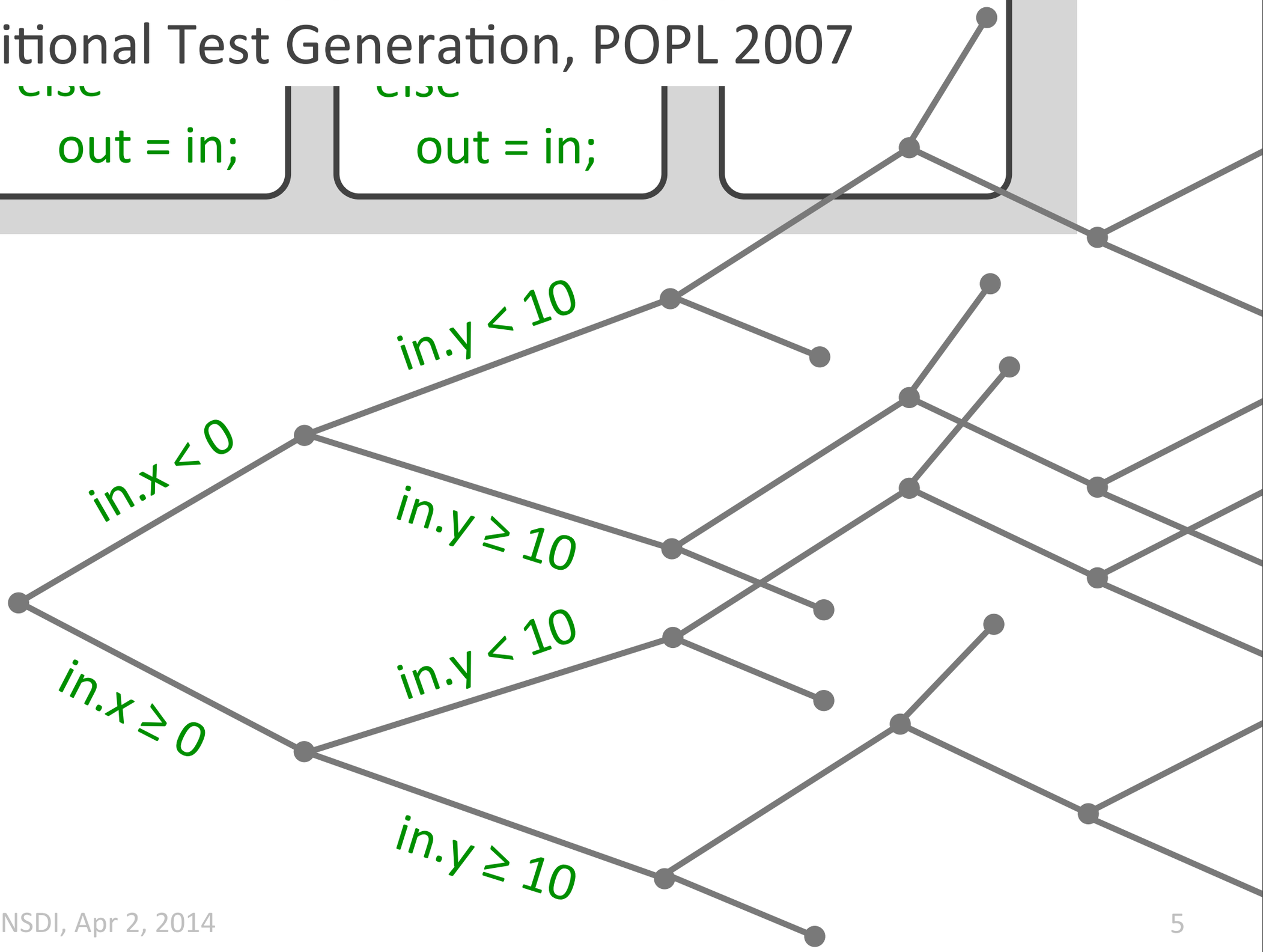
```
if (in.y < 10)
   out = ...;
else
   out = in;
```



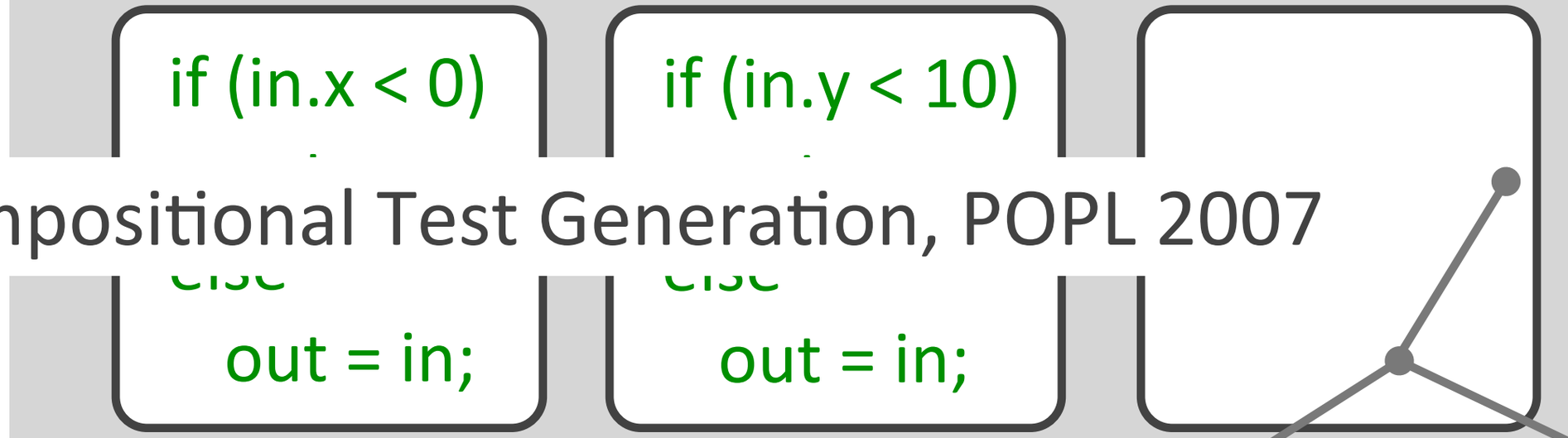
Compositional Test Generation, POPL 2007

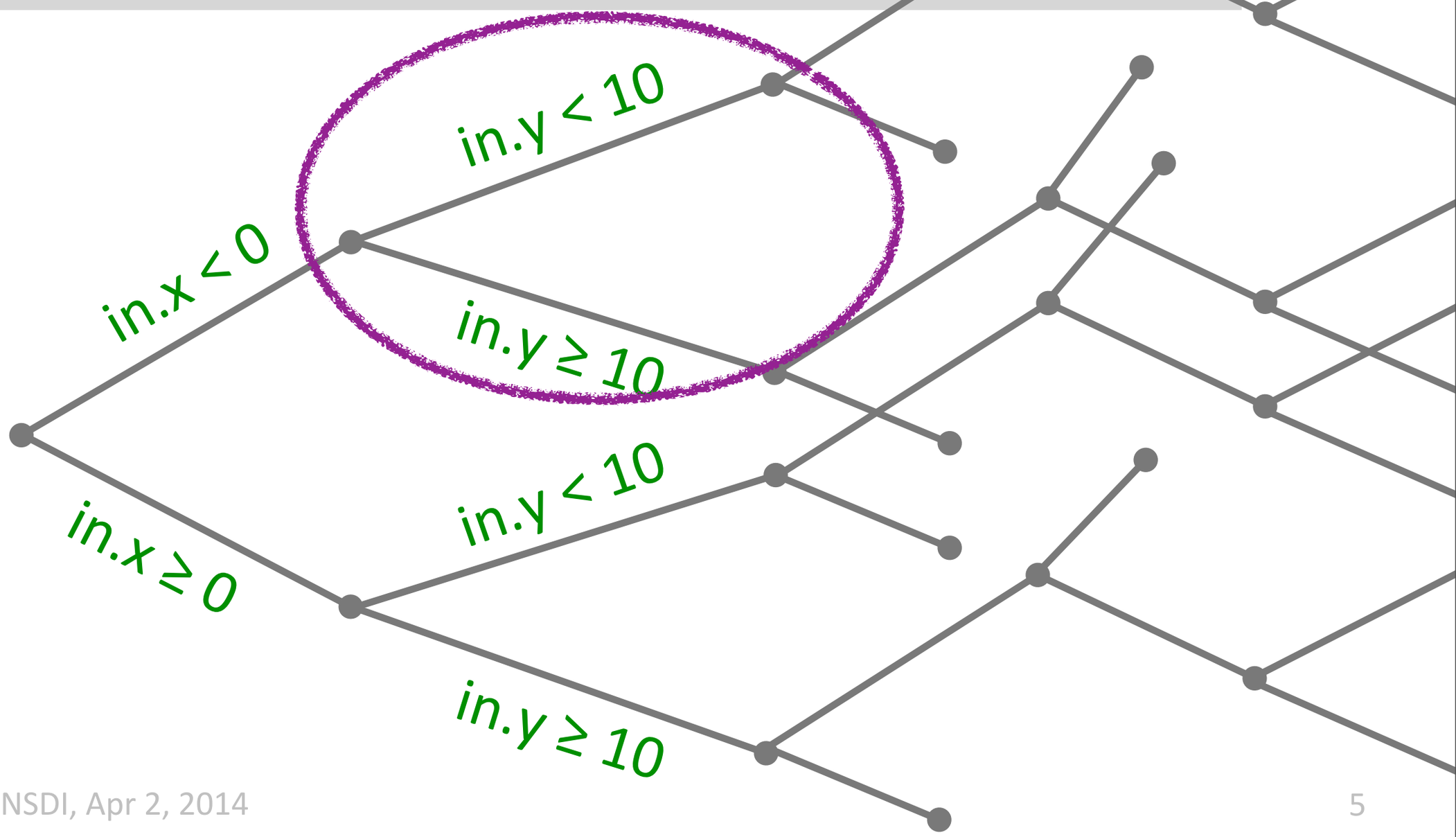
```
if (in.x < 0)
  out = in;
```

```
if (in.y < 10)
  out = in;
```



Compositional Test Generation, POPL 2007

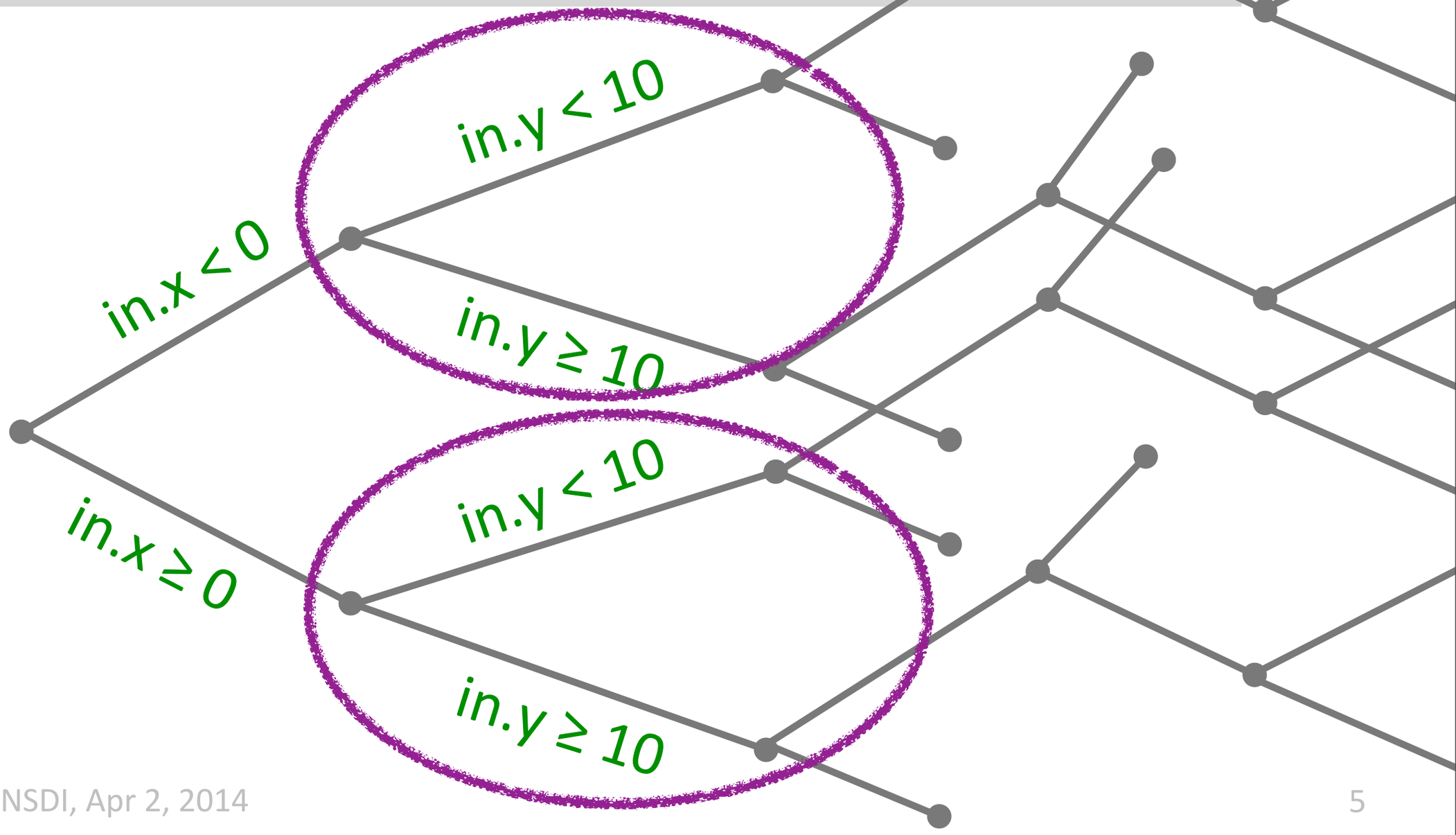




Compositional Test Generation, POPL 2007

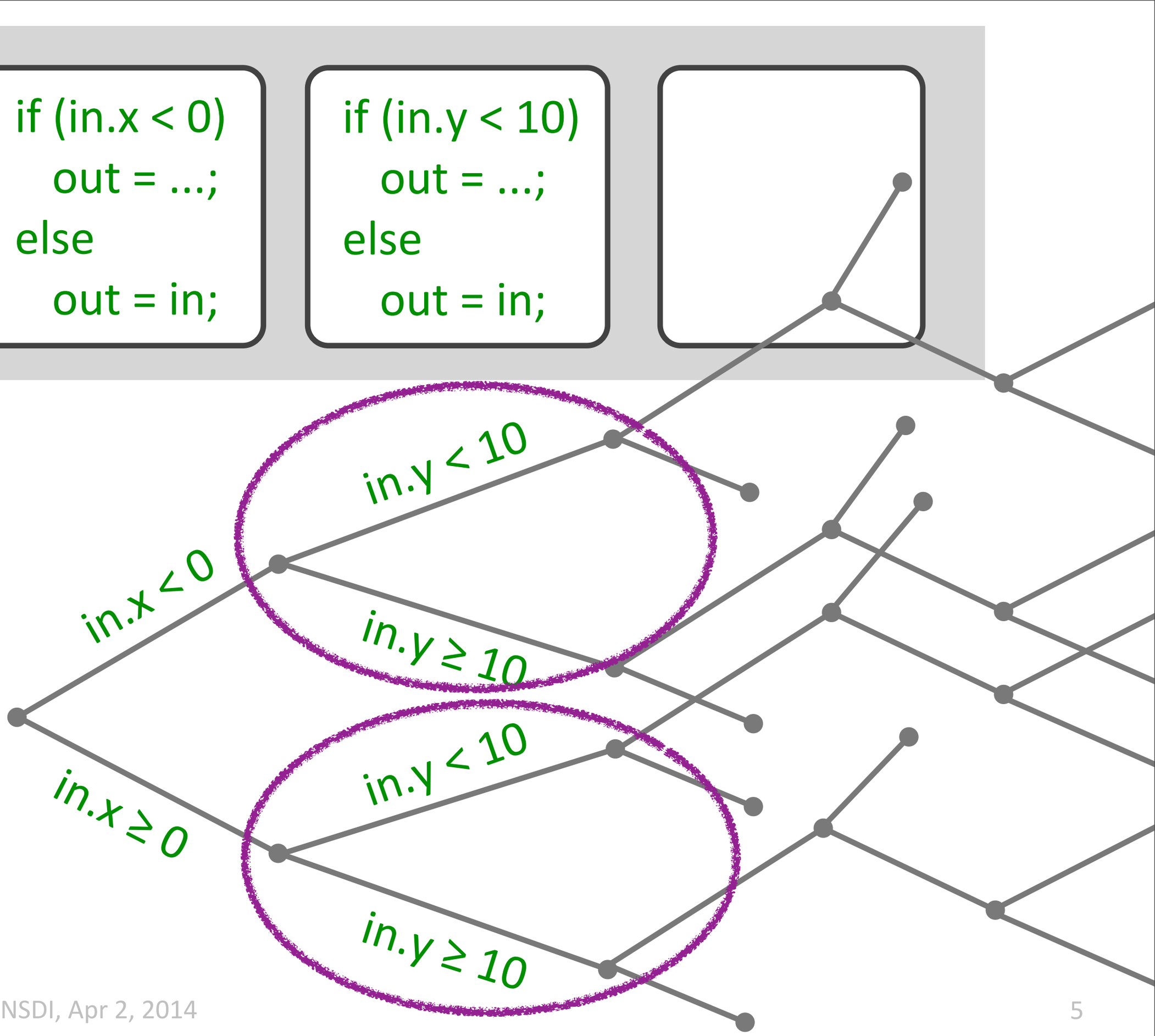
```
if (in.x < 0)
  out = in;
else
  out = in;
```

```
if (in.y < 10)
  out = in;
else
  out = in;
```



```
if (in.x < 0)
    out = ...;
else
    out = in;
```

```
if (in.y < 10)
    out = ...;
else
    out = in;
```



Dataplane-specific verification

- Define the domain
  - *propose rules on how to write dataplanes*
  - *make it easy to apply composition*

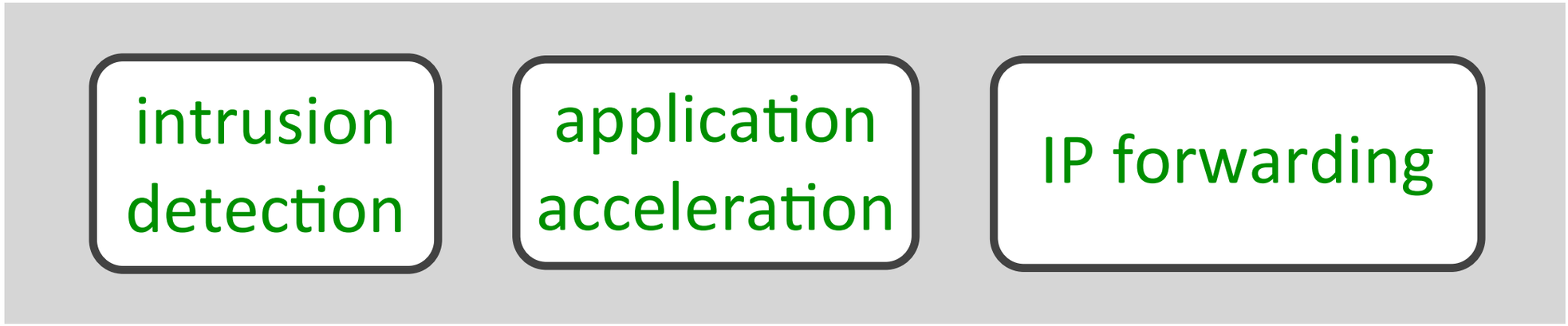
- Leverage the domain specificity
  - *use it to sidestep path explosion*
  - *open the door to dataplane verification*

Outline

- Pipeline
- Loops
- Data structures
- Results

Outline

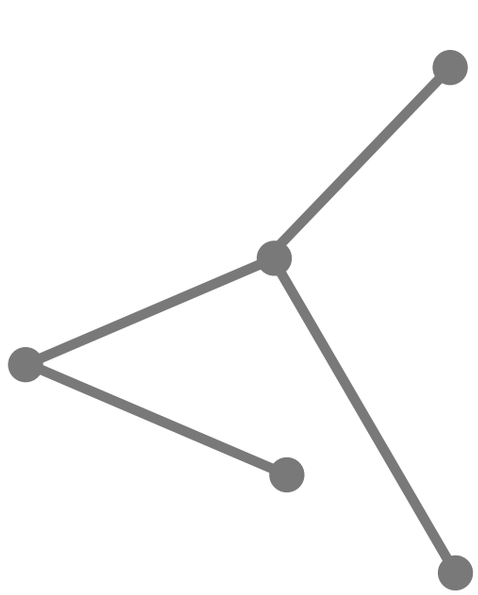
- Pipeline
- Loops
- Data structures
- Results


intrusion detection
application acceleration
IP forwarding

intrusion detection

application acceleration

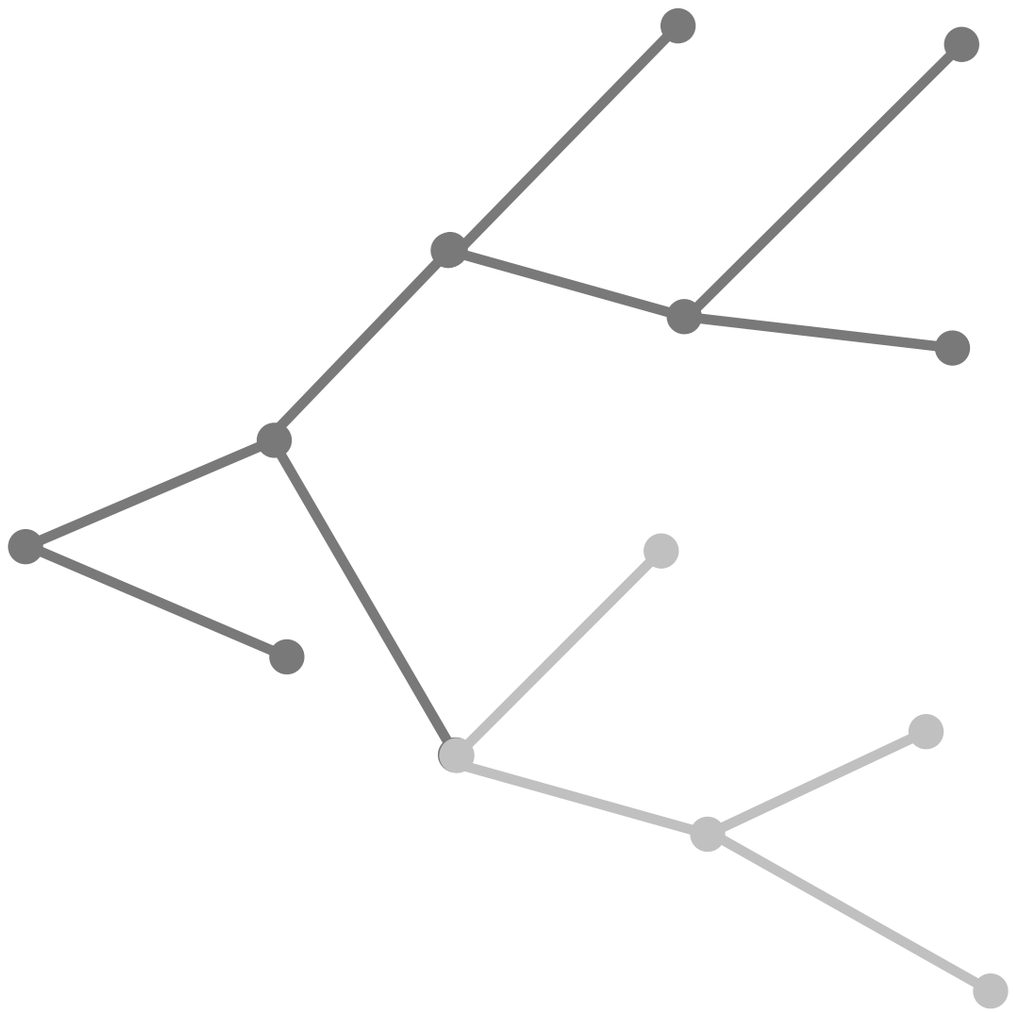
IP forwarding



intrusion detection

application acceleration

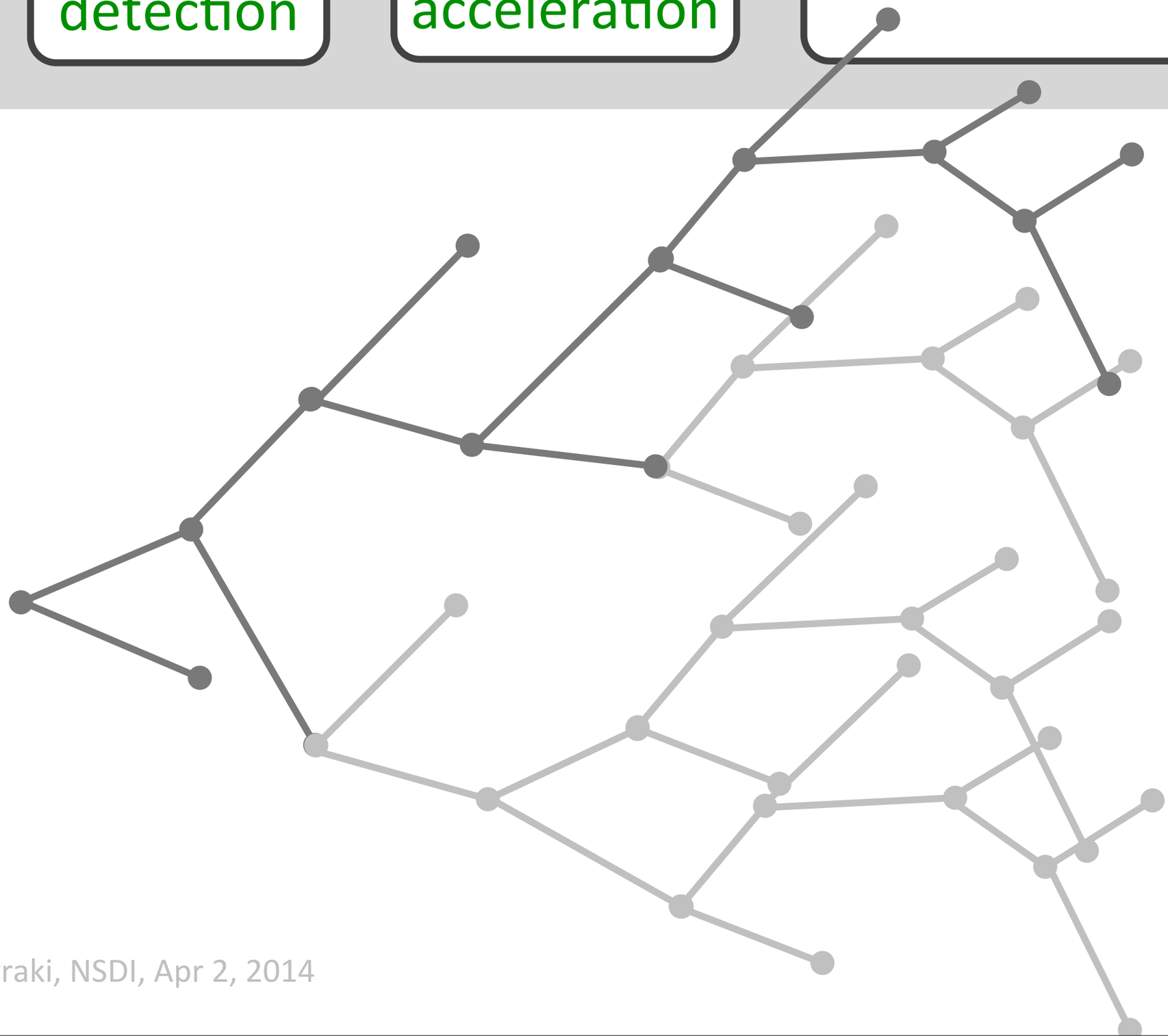
IP forwarding



intrusion detection

application acceleration

IP forwarding

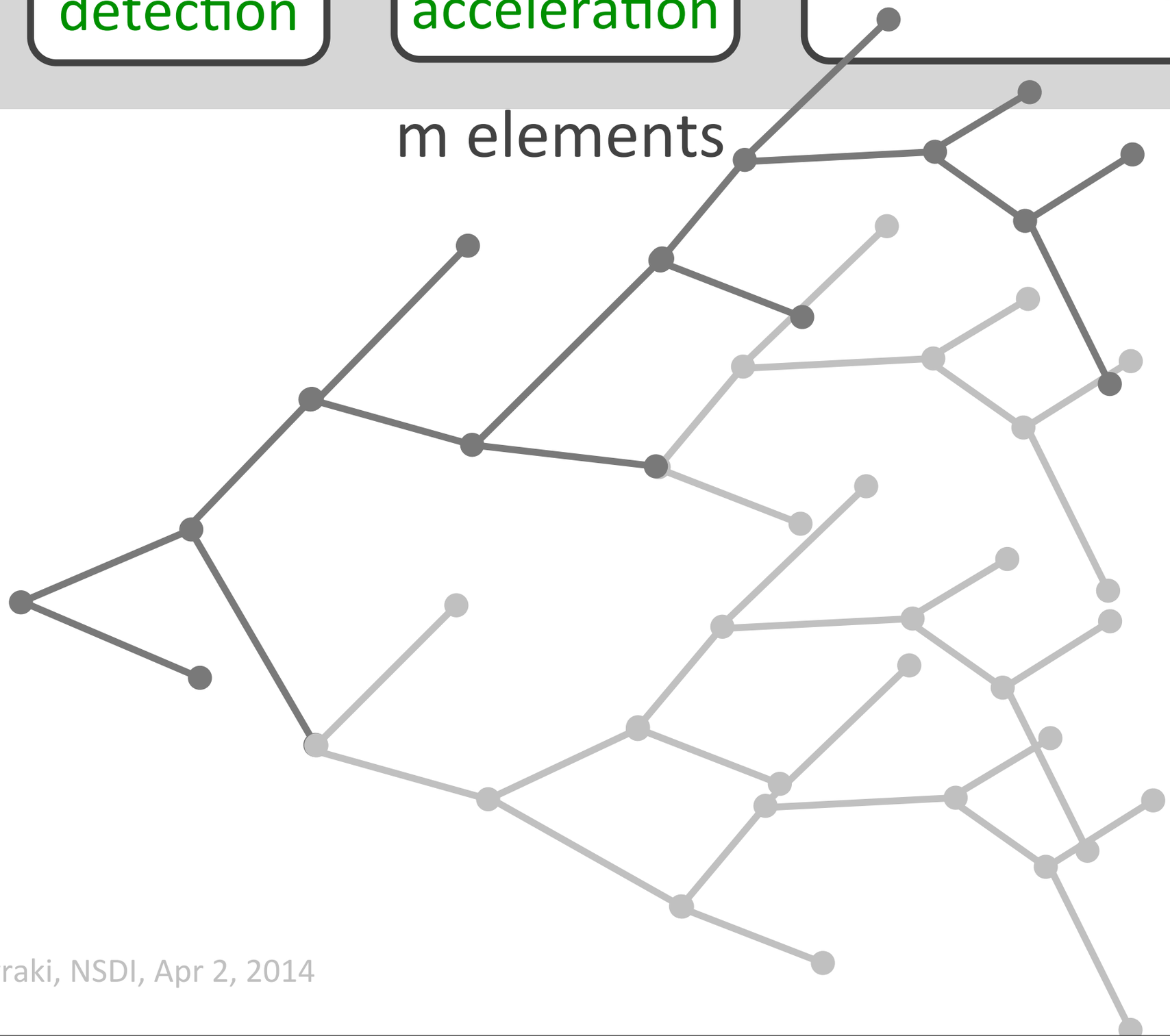


intrusion detection

application acceleration

IP forwarding

m elements



intrusion detection

application acceleration

IP forwarding

m elements

n branches per element

intrusion detection

application acceleration

IP forwarding

m elements

n branches per element

verification time ~ $2^{n\,m}$

intrusion detection

application acceleration

IP forwarding

m elements

n branches per element

intrusion detection

application acceleration

IP forwarding

m elements

do not share mutable state

n branches per element

intrusion detection

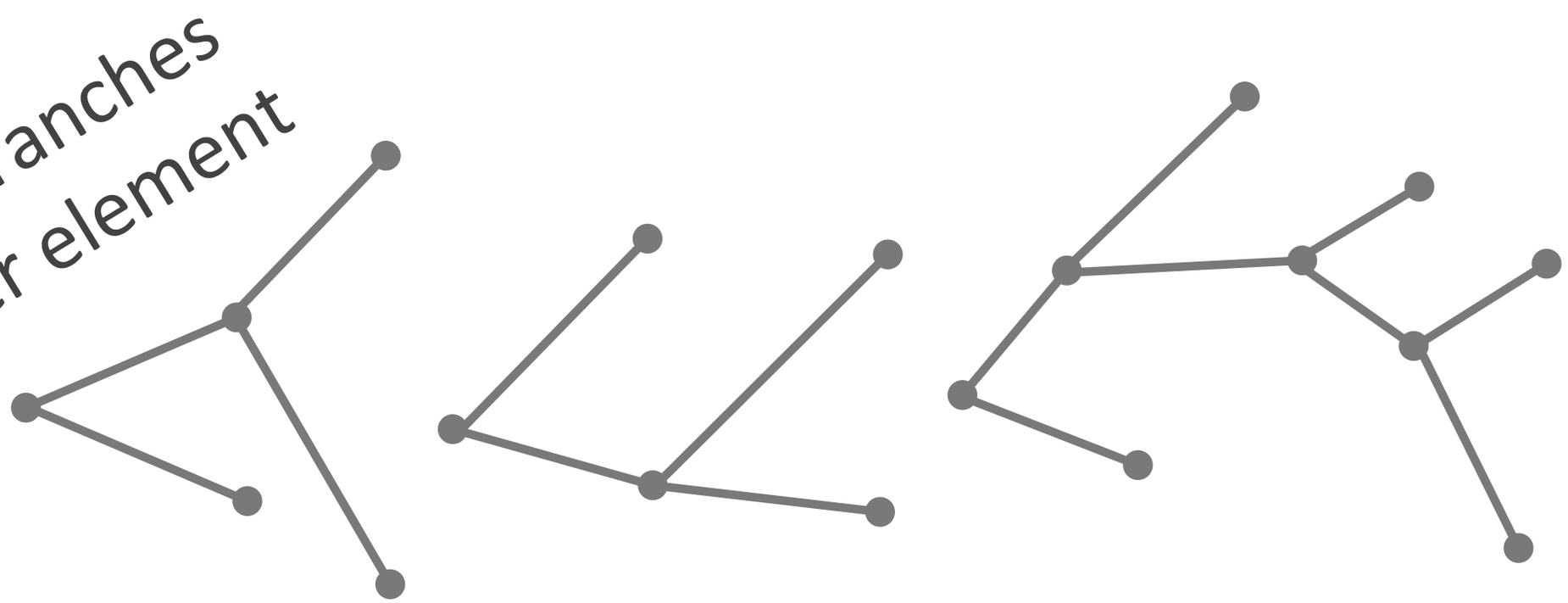
application acceleration

IP forwarding

m elements

do not share mutable state

n branches per element



intrusion detection

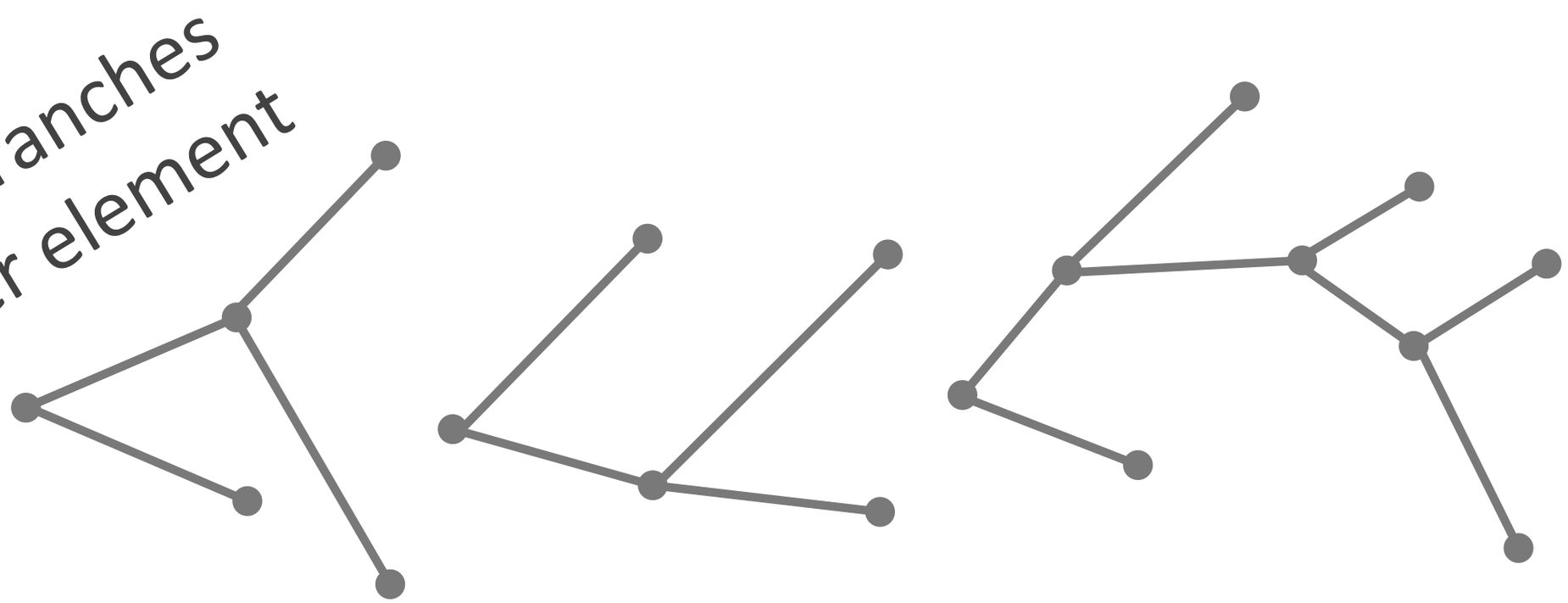
application acceleration

IP forwarding

m elements

do not share mutable state

n branches per element



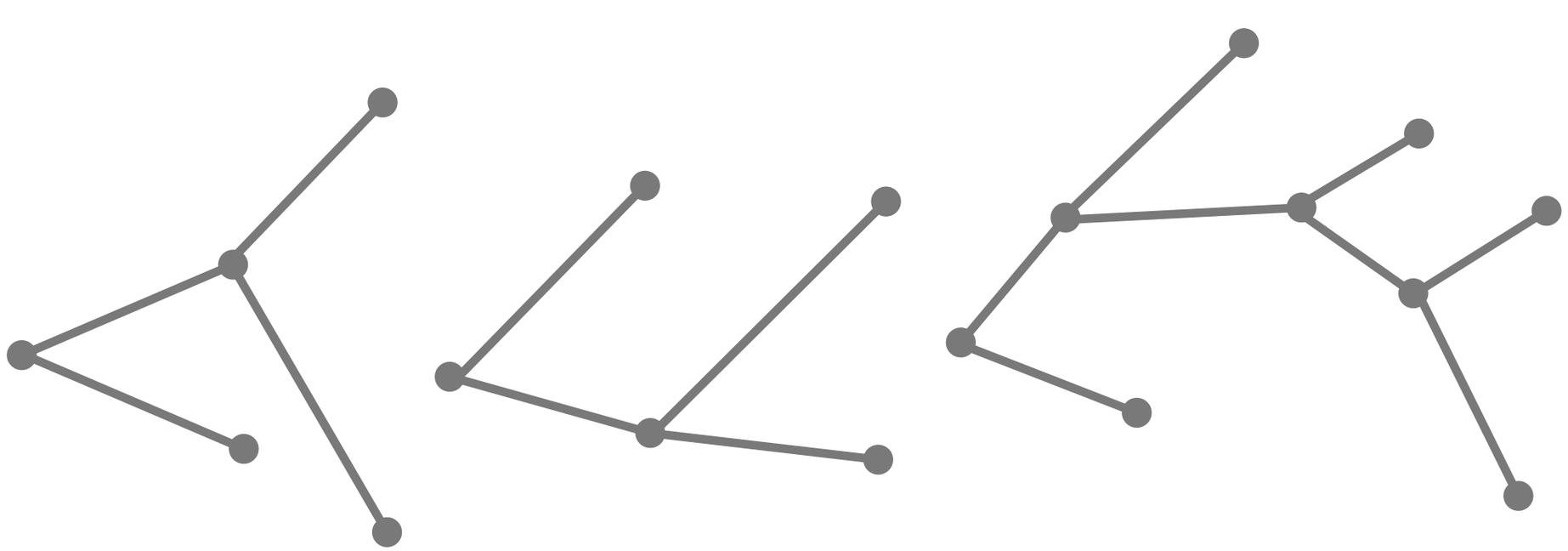
verification time ~ m 2^n

intrusion detection

application acceleration

IP forwarding

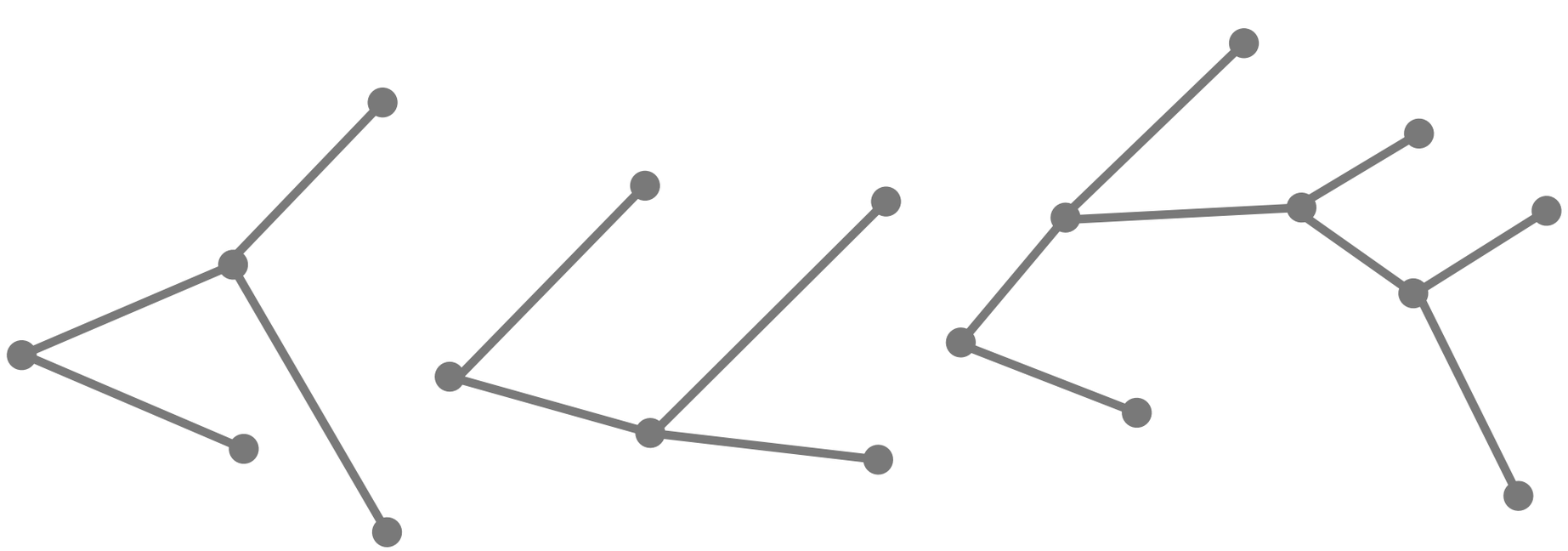
do not share mutable state



intrusion detection

application acceleration

do not share mutable state

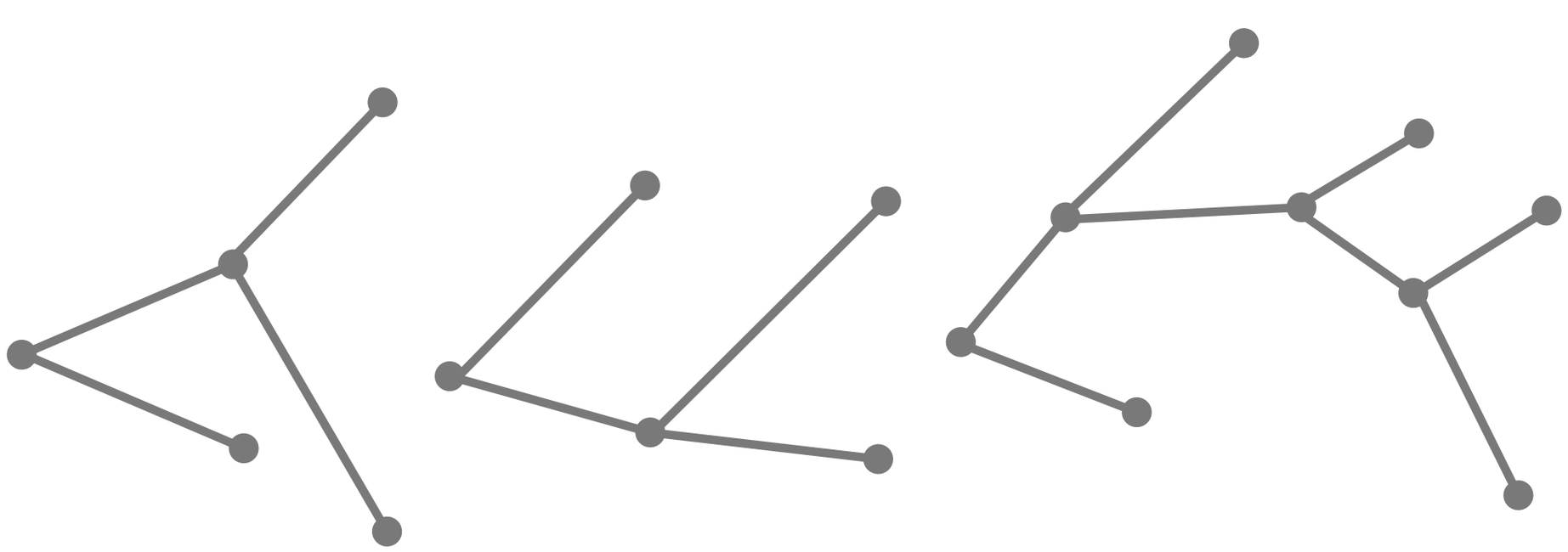


intrusion detection

application acceleration

```
...
assert(src != dst);
...
```

do not share mutable state

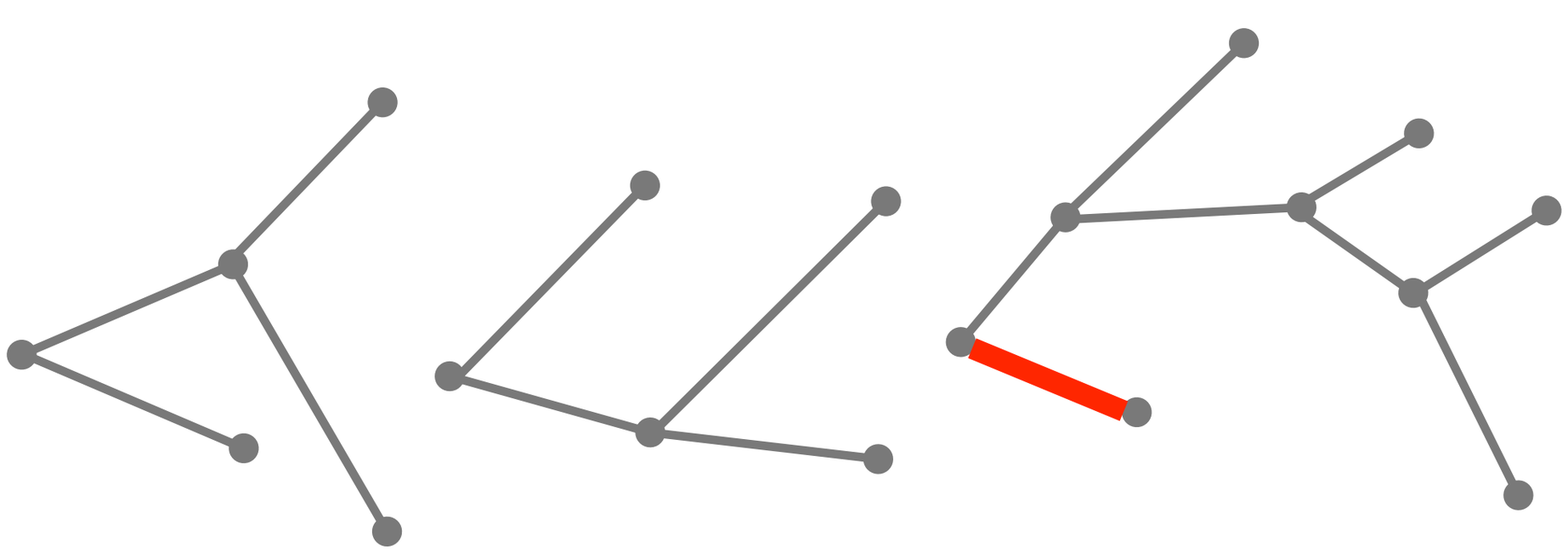


intrusion detection

application acceleration

```
...
assert(src != dst);
...
```

do not share mutable state

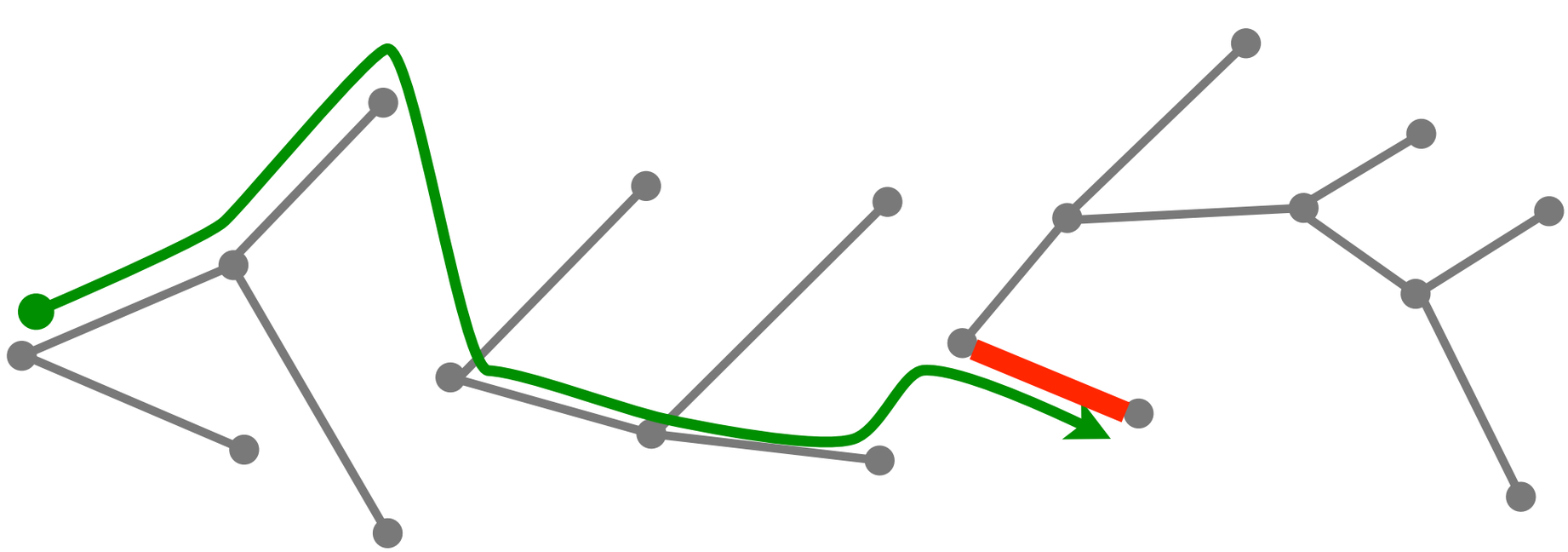


intrusion detection

application acceleration

```
...
assert(src != dst);
...
```

do not share mutable state



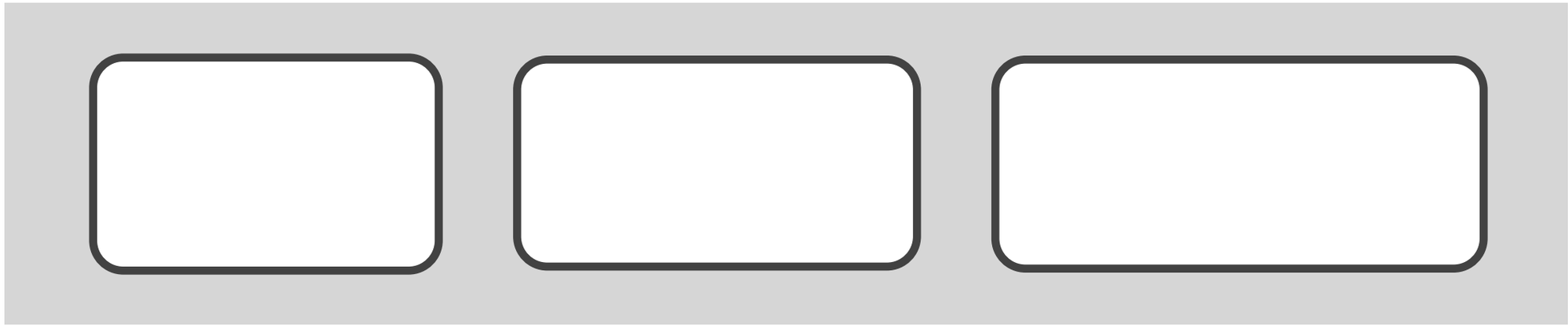
Pipeline decomposition

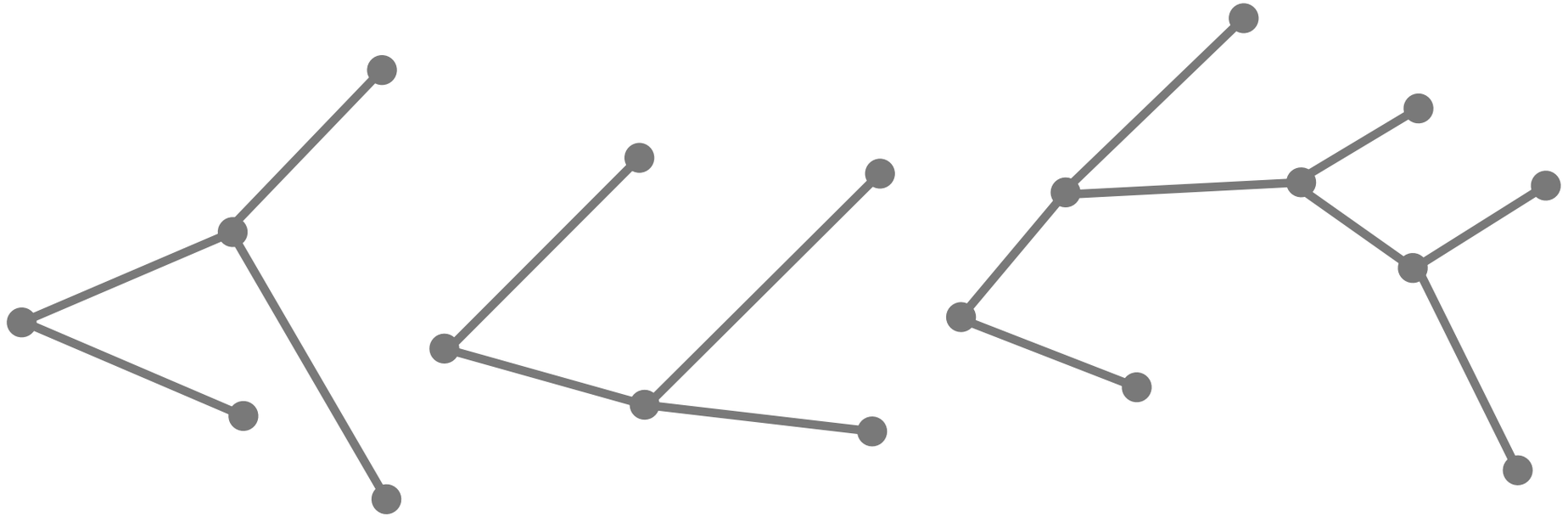
- Rule: pipeline structure
  - *distinct packet-processing elements*
  - *do not share mutable state*

- Effect: compose at the element level
  - *can reduce #paths from* $\sim 2^{n\,m}$
  - *to* $\sim m\,2^{n}$

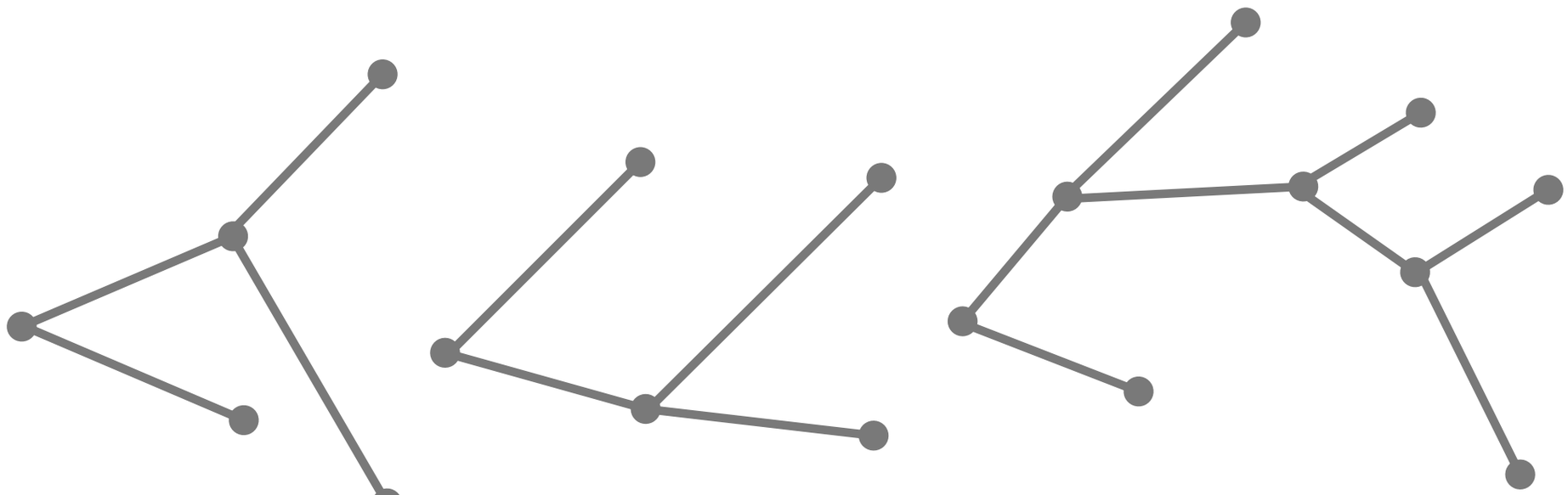
Outline

- Pipeline
- Loops
- Data structures
- Results

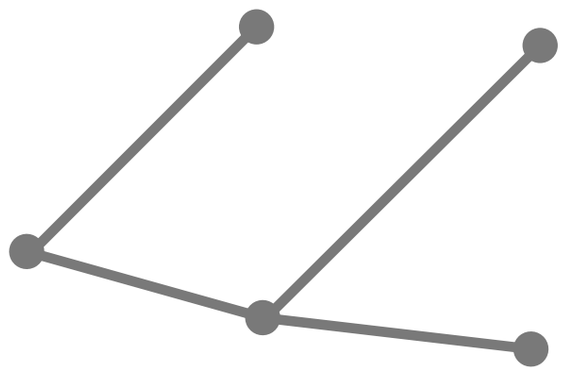





IP options

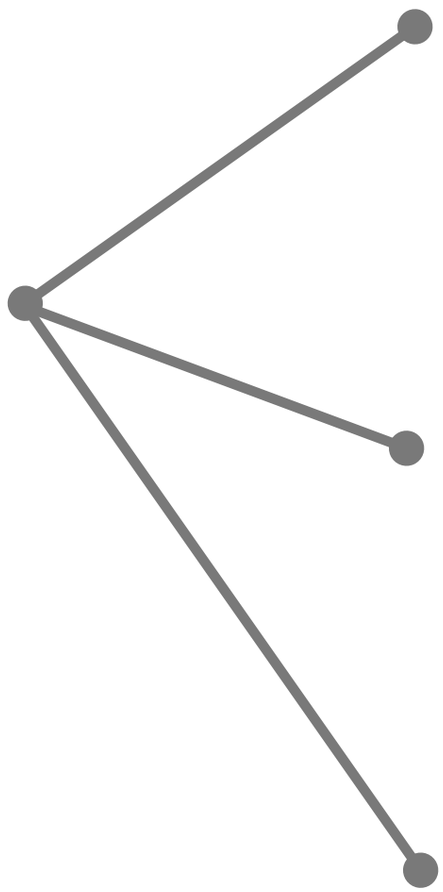


IP options

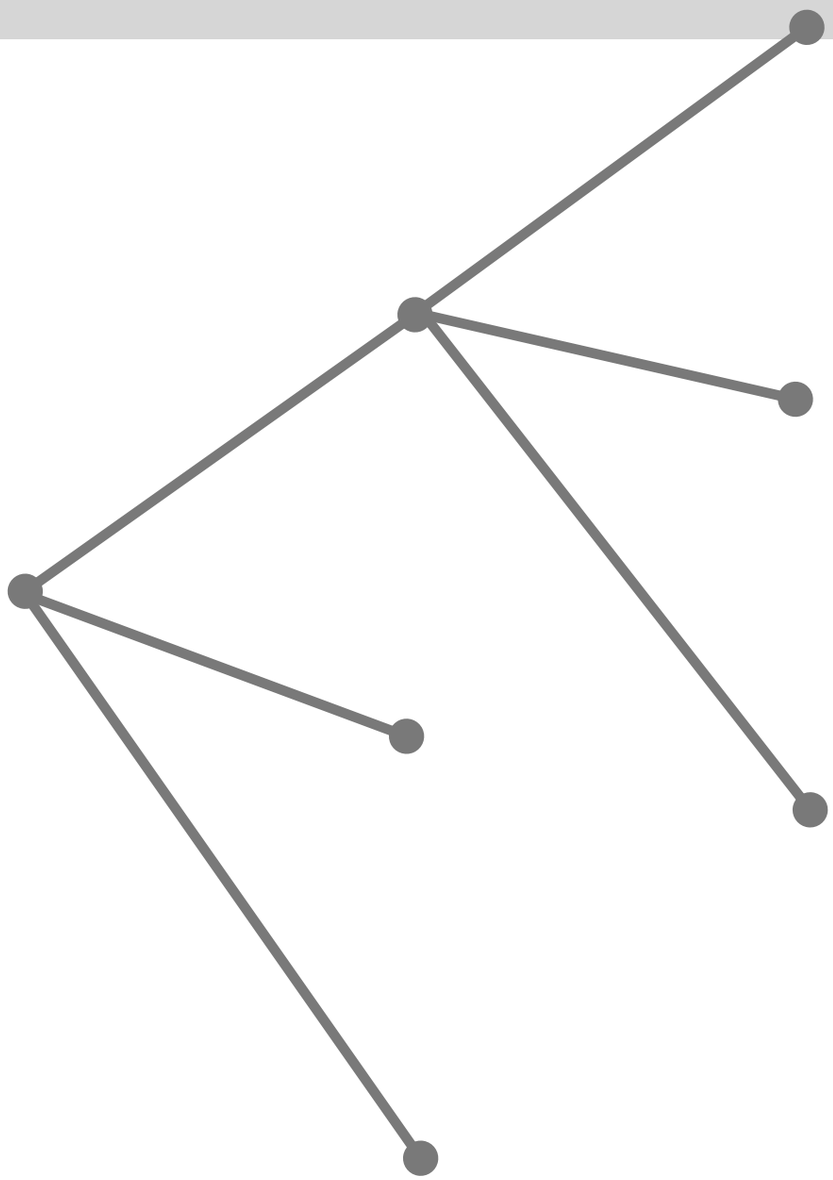


option #1 option #2 ... option #m

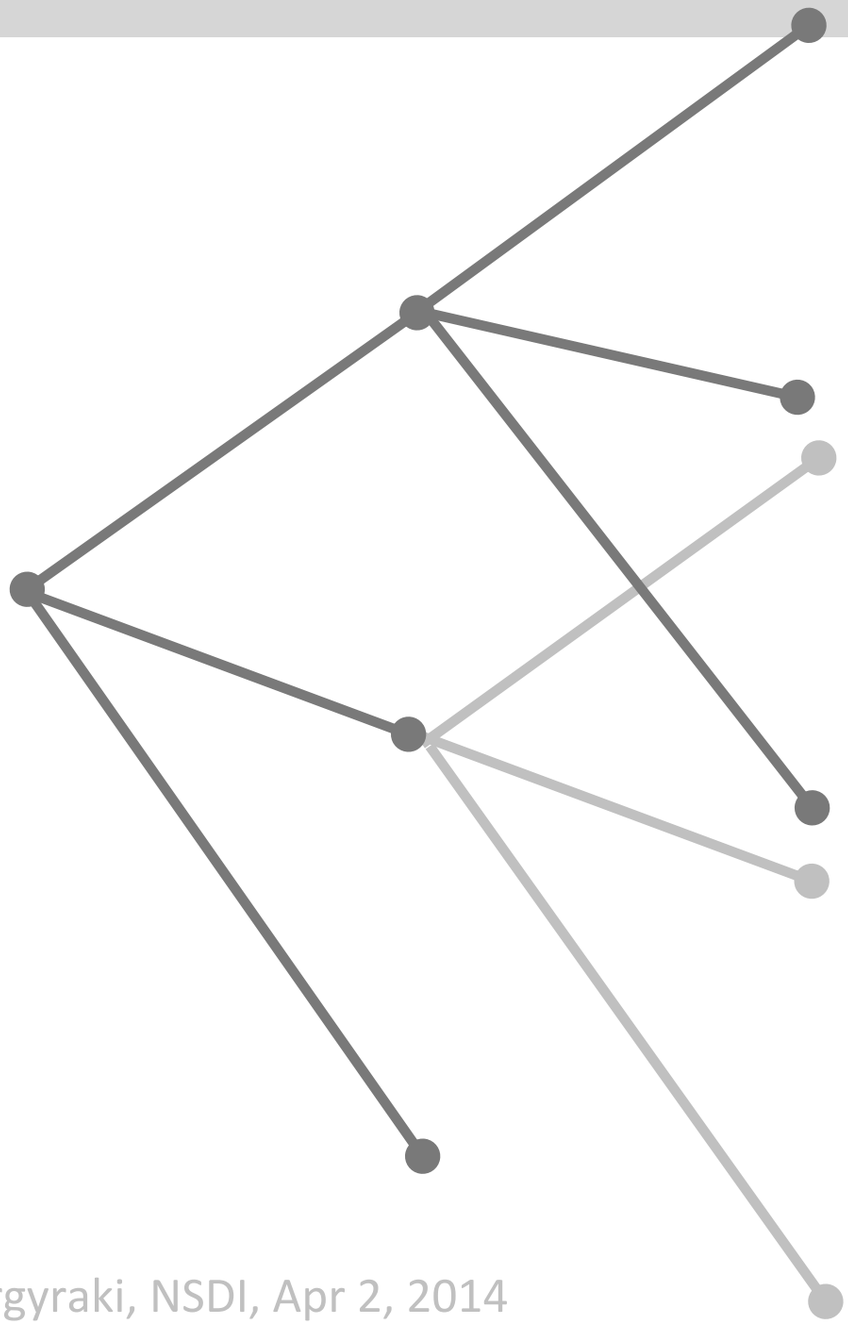
option #1 option #2 ... option #m



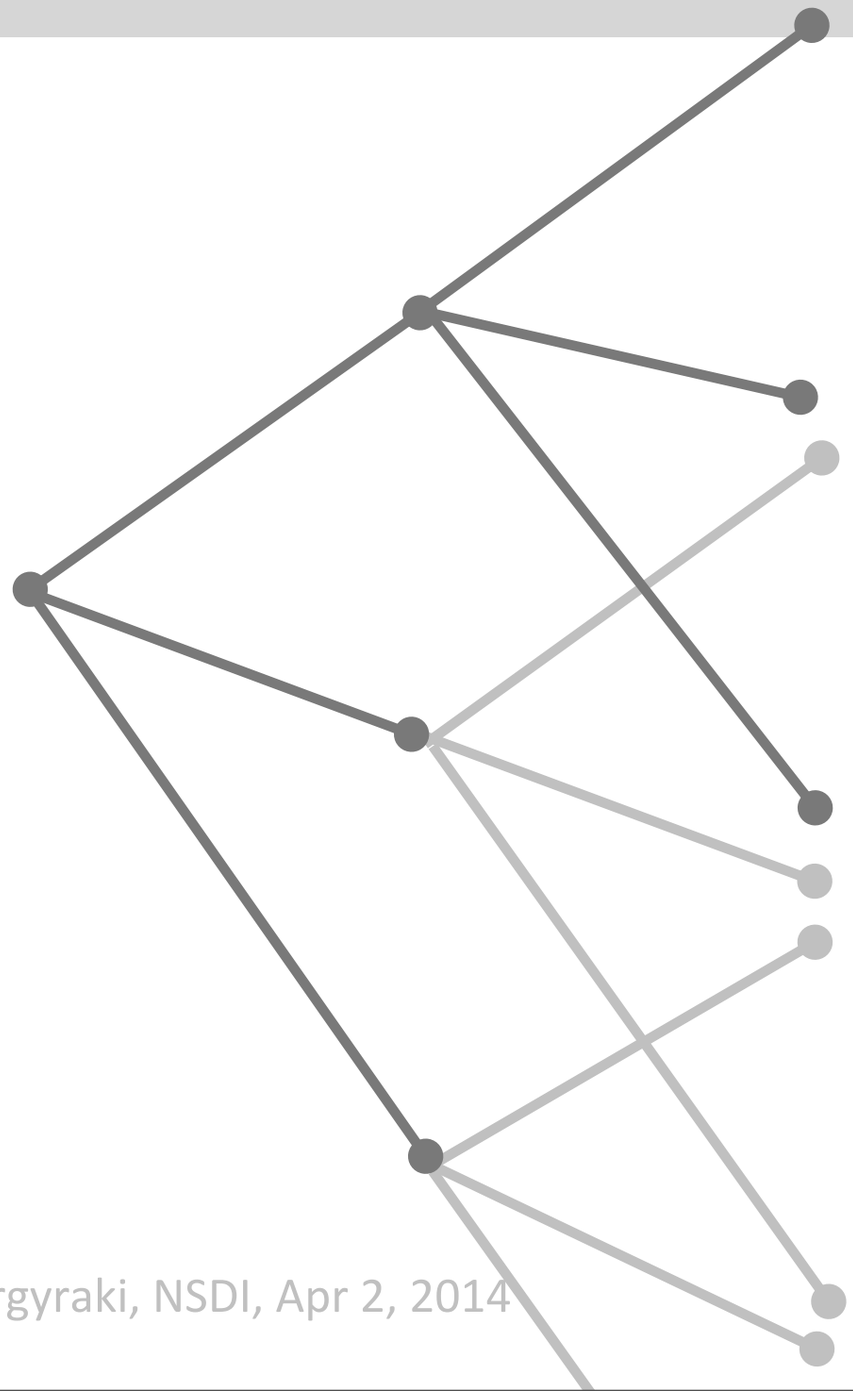
option #1 option #2 ... option #m



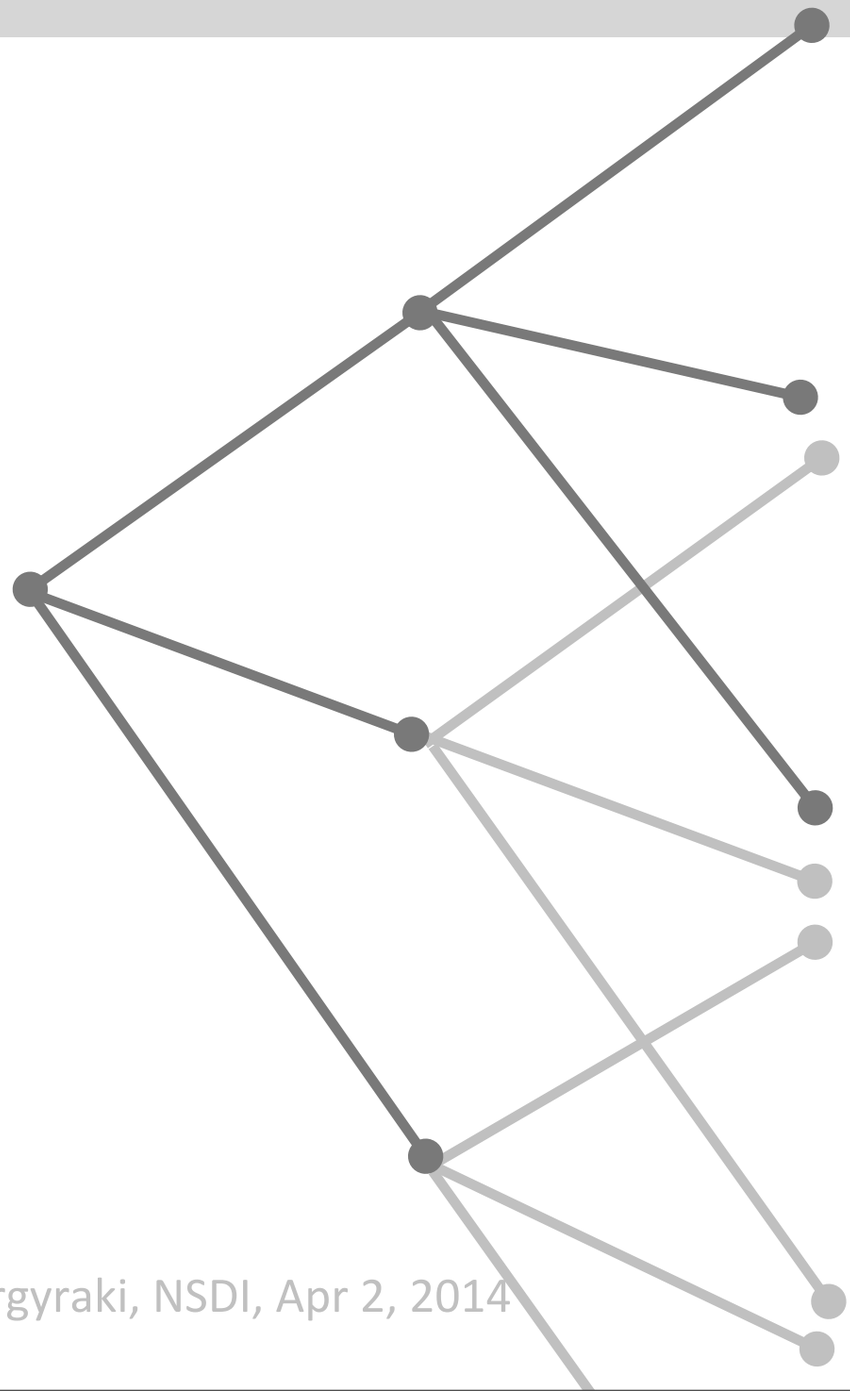
option #1 option #2 ... option #m



option #1 option #2 ... option #m

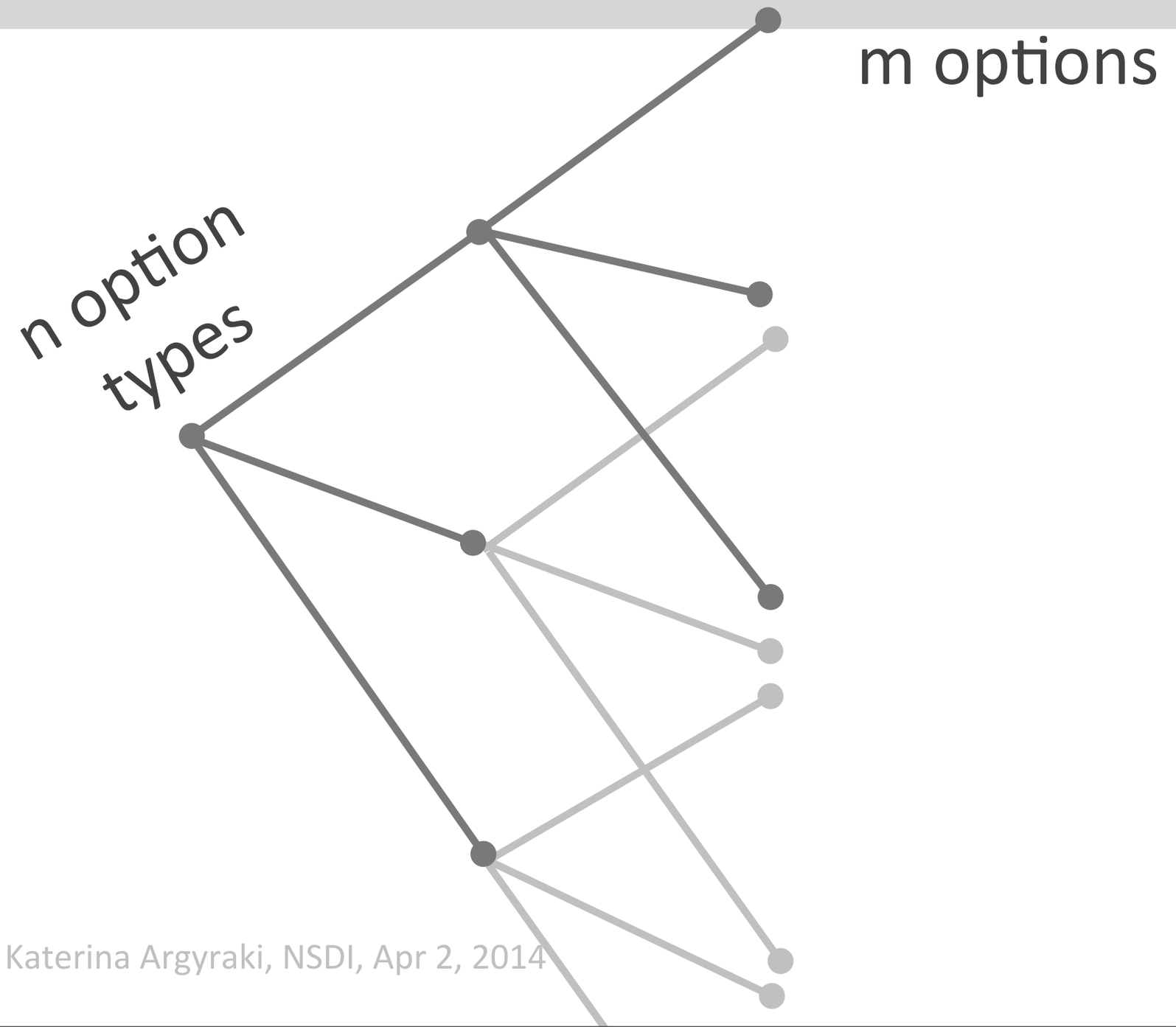


option #1 option #2 ... option #m

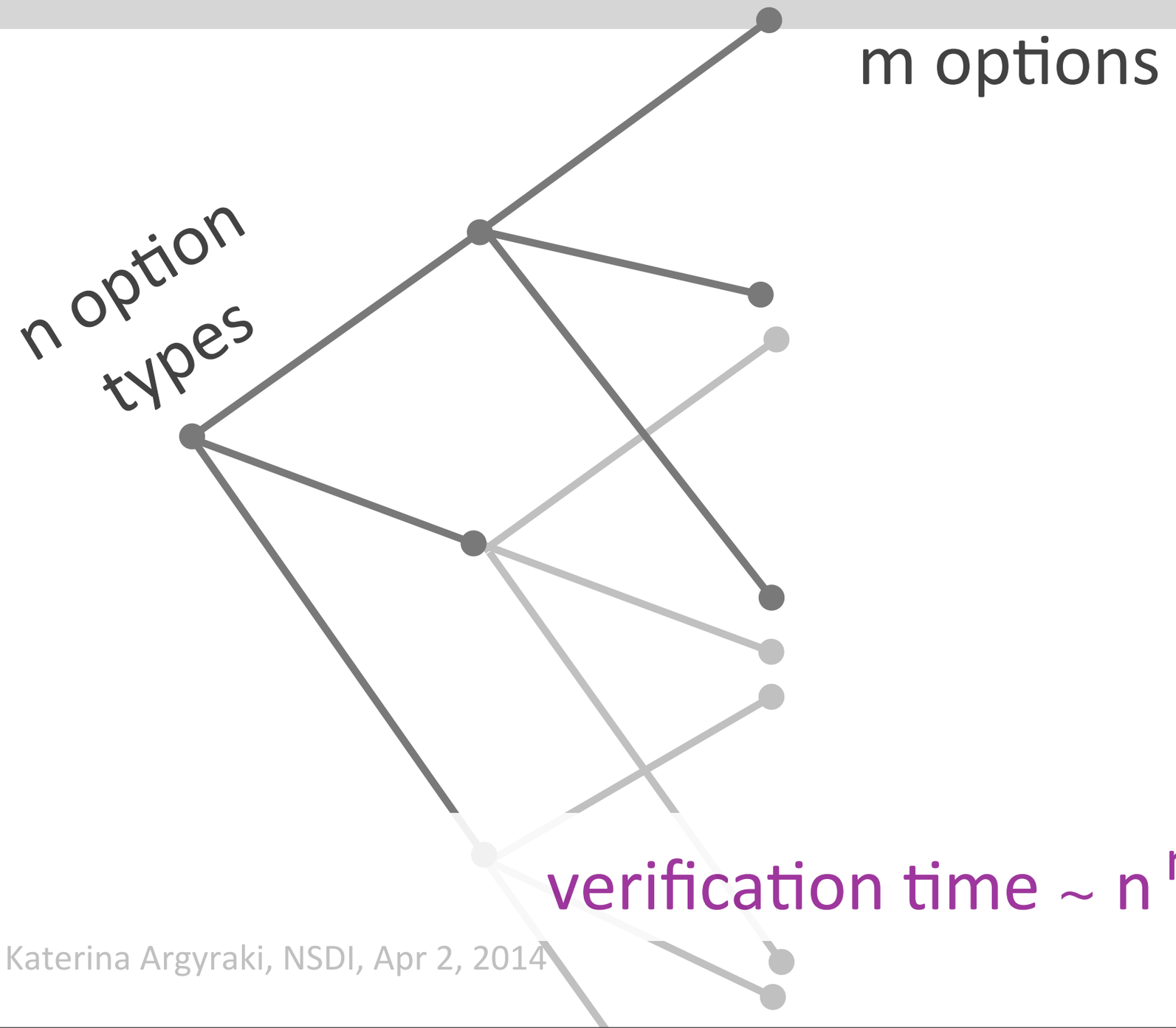


m options

option #1 option #2 ... option #m



option #1 option #2 ... option #m



verification time ~ n^m

option #1 | option #2 | ... | option #m

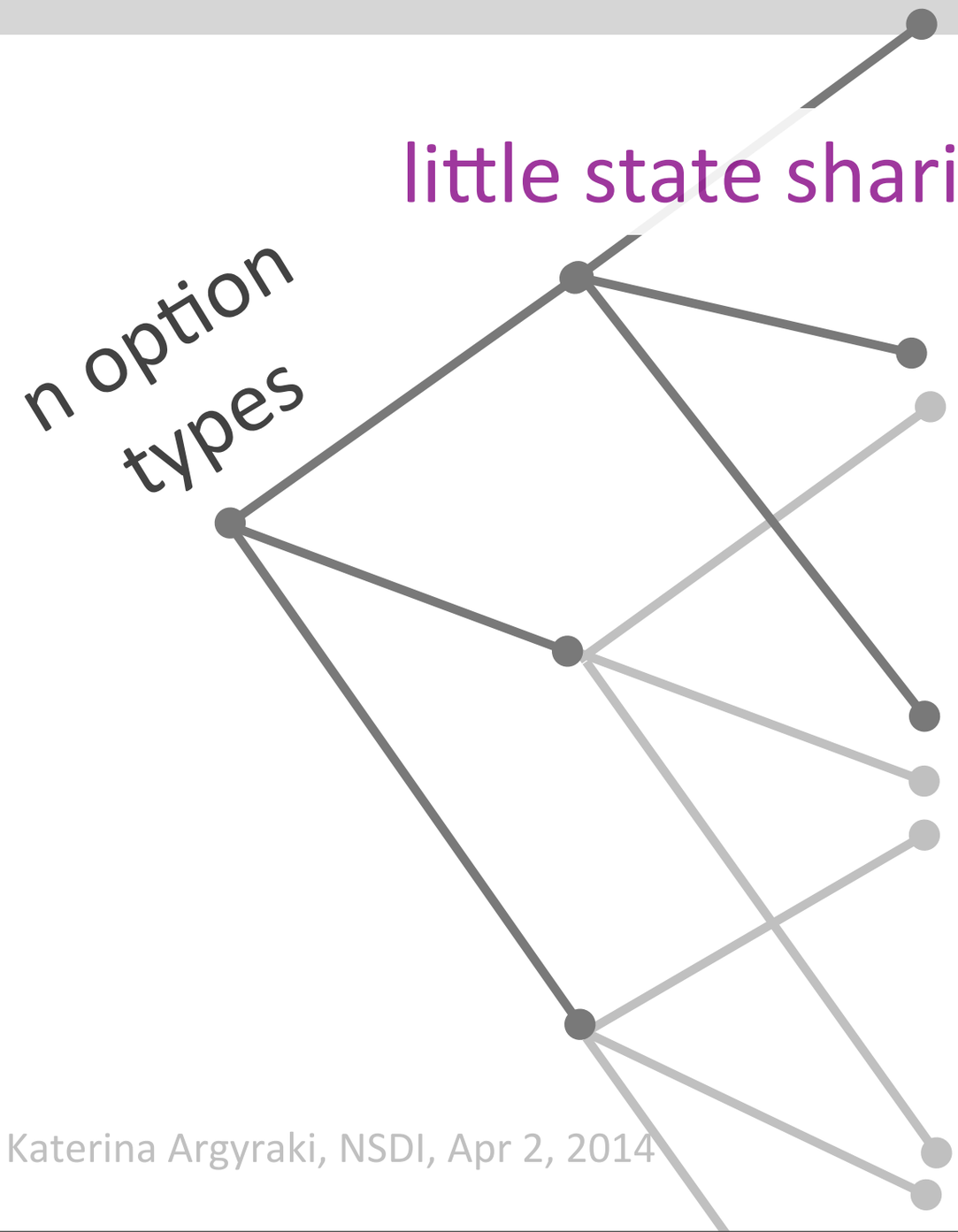
m options

n option types

option #1 | option #2 | ... | option #m

m options

little state sharing across iterations

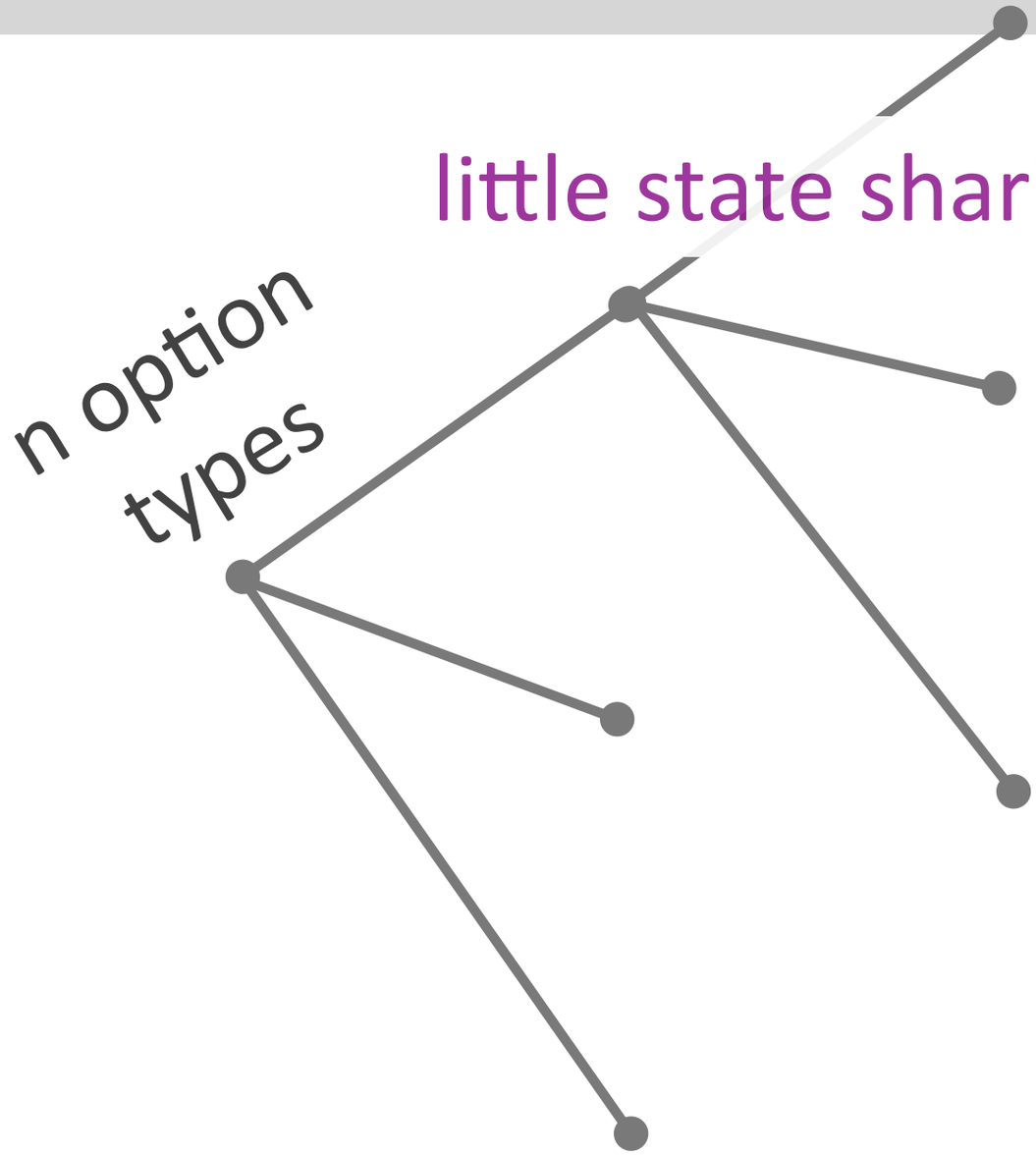


option #1 | option #2 | ... | option #m

m options

little state sharing across iterations

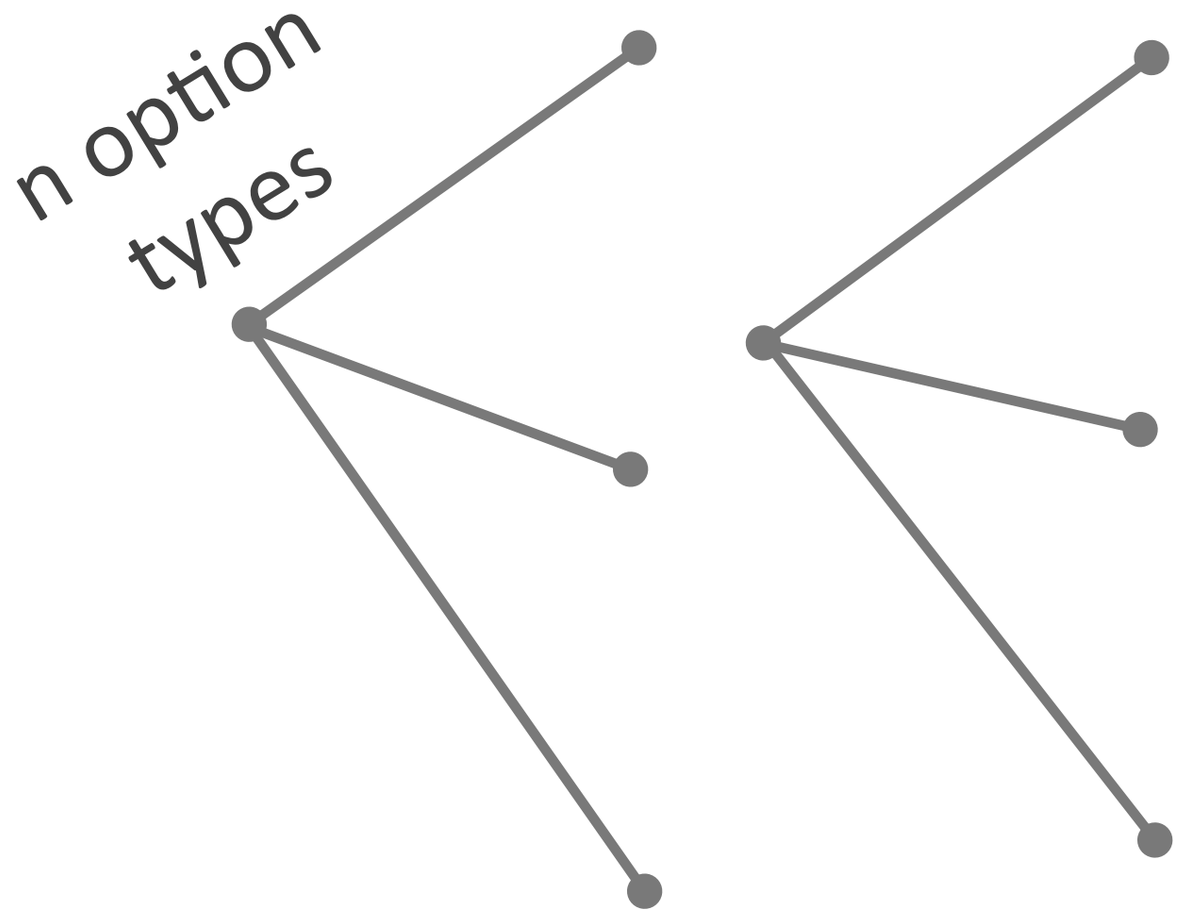
n option types



option #1 | option #2 | ... | option #m

m options

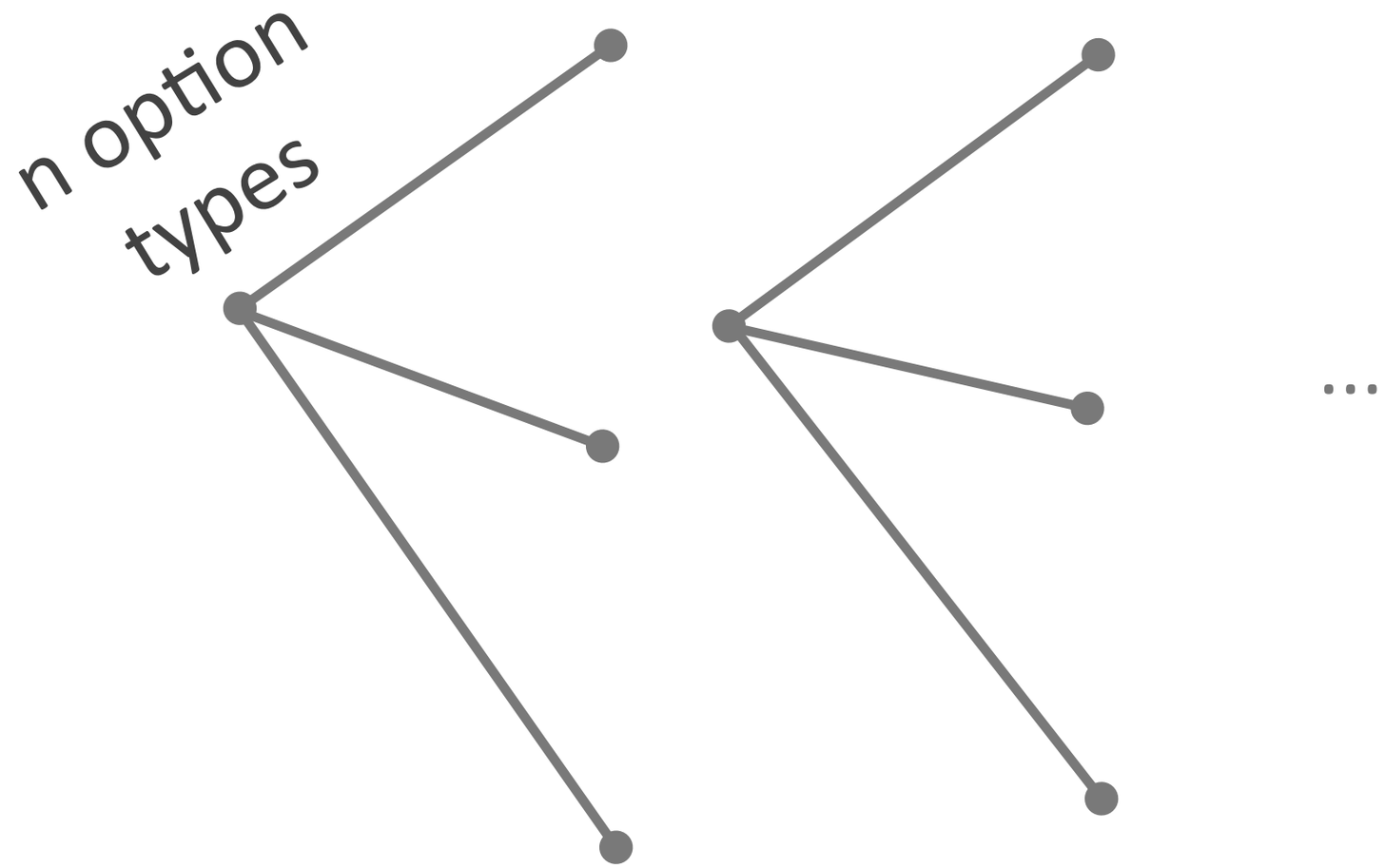
little state sharing across iterations



option #1 | option #2 | ... | option #m

m options

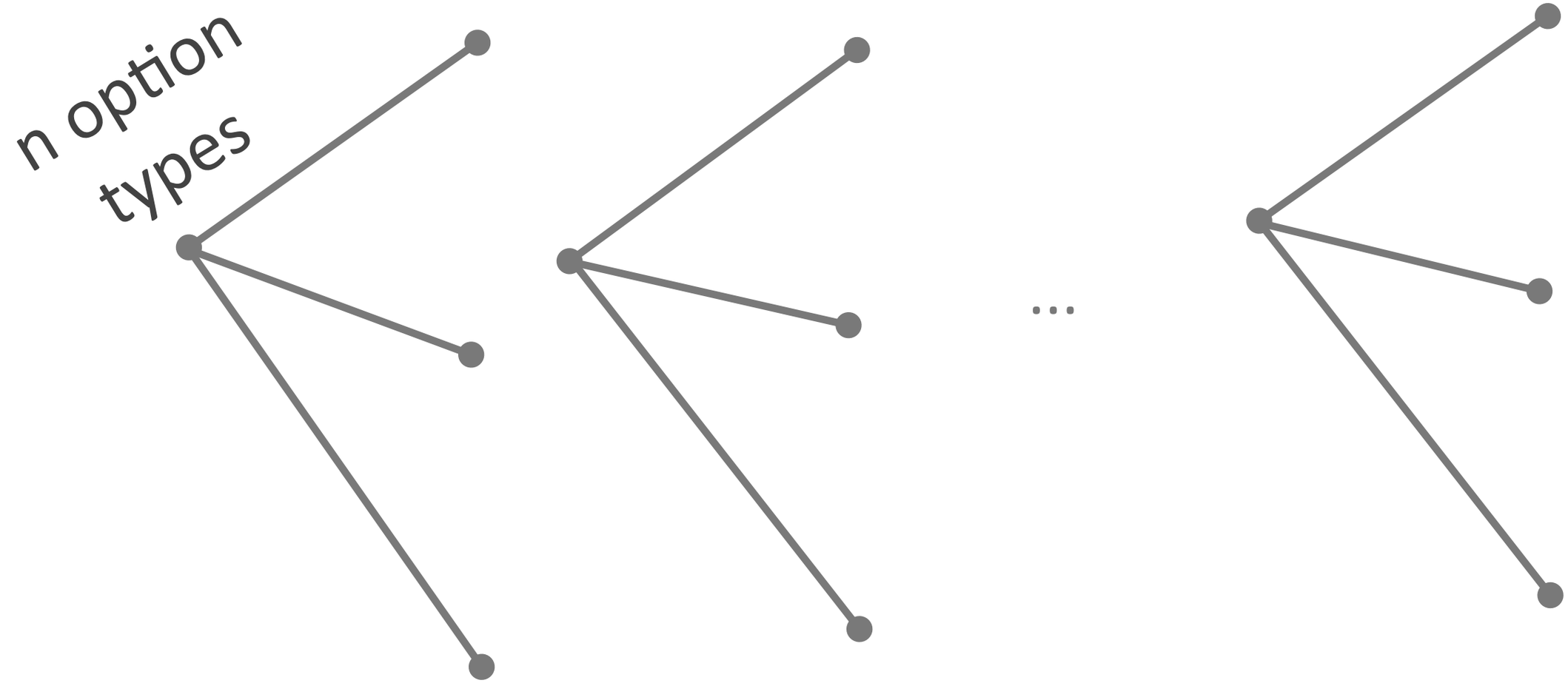
little state sharing across iterations



option #1 | option #2 | ... | option #m

m options

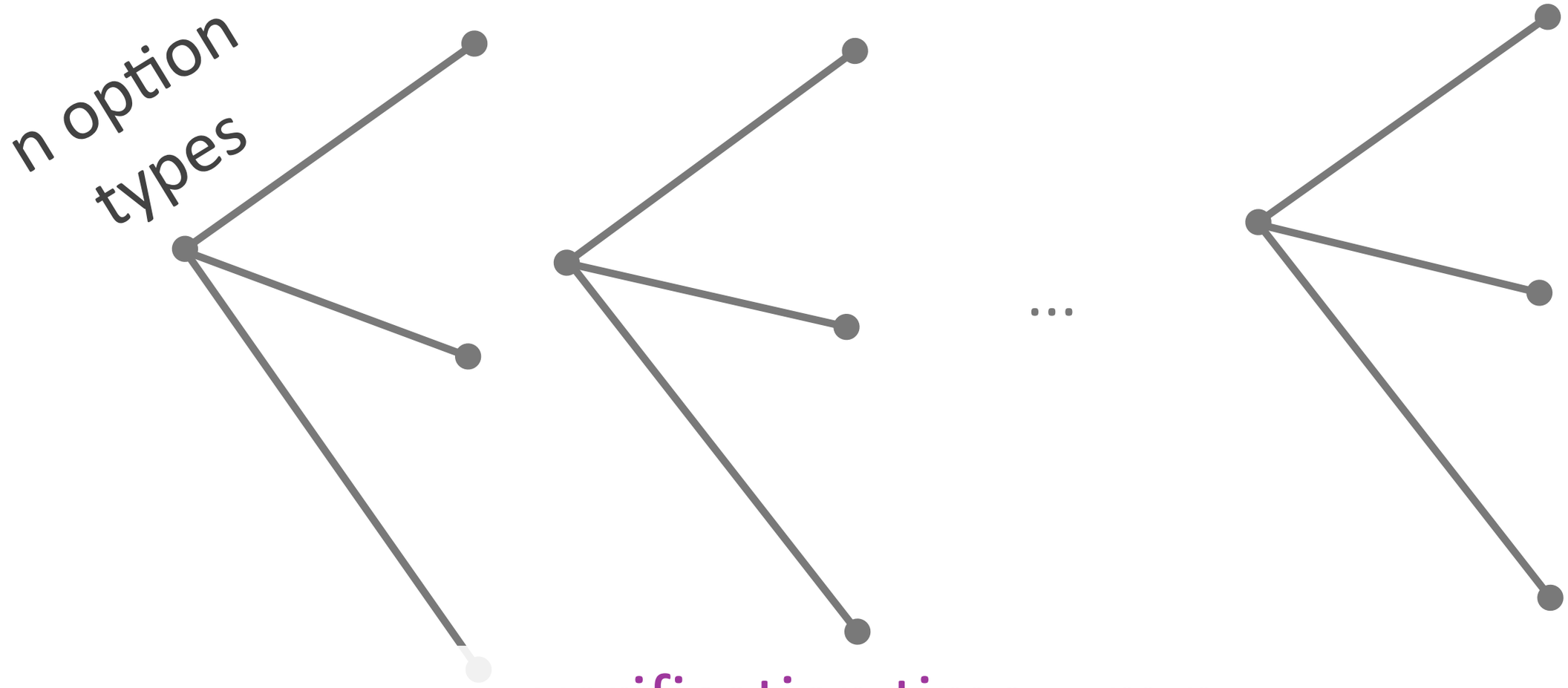
little state sharing across iterations



option #1 | option #2 | ... | option #m

m options

little state sharing across iterations



verification time ~ m n

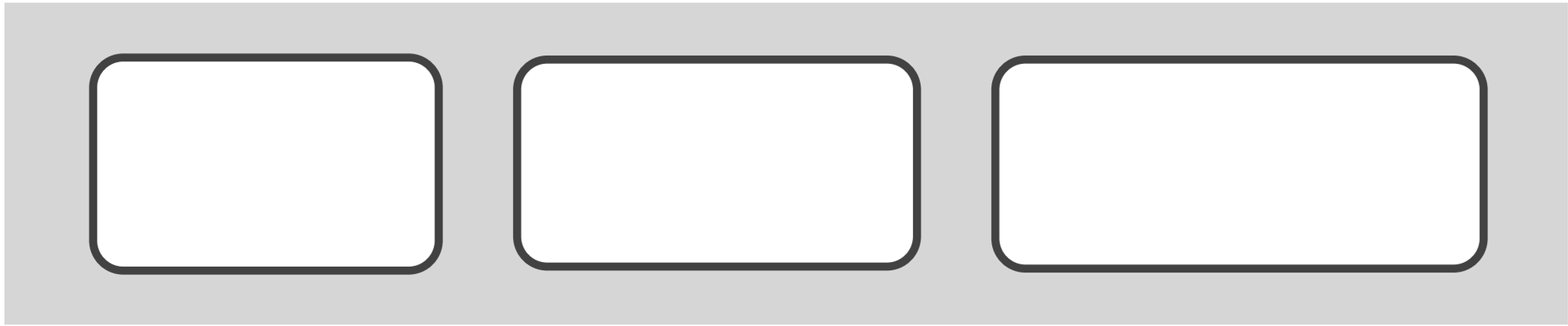
Loop decomposition

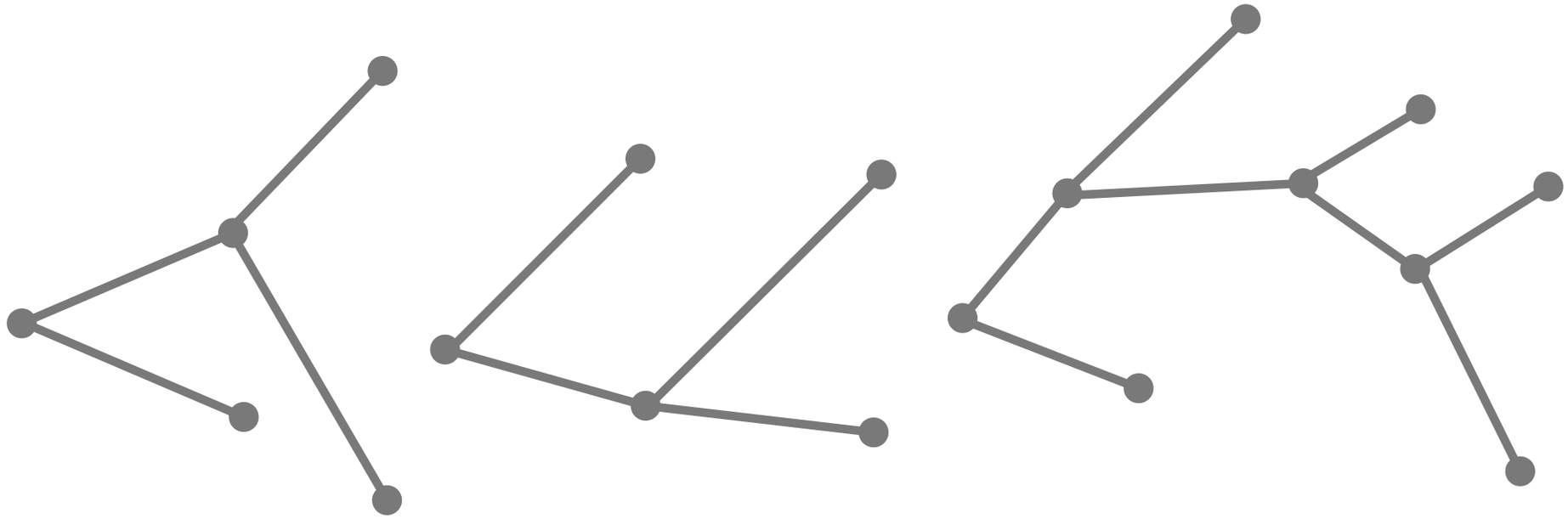
- Rule: "mini-pipeline" structure
  - *little state shared across iterations*
  - *made explicit by the programmer*

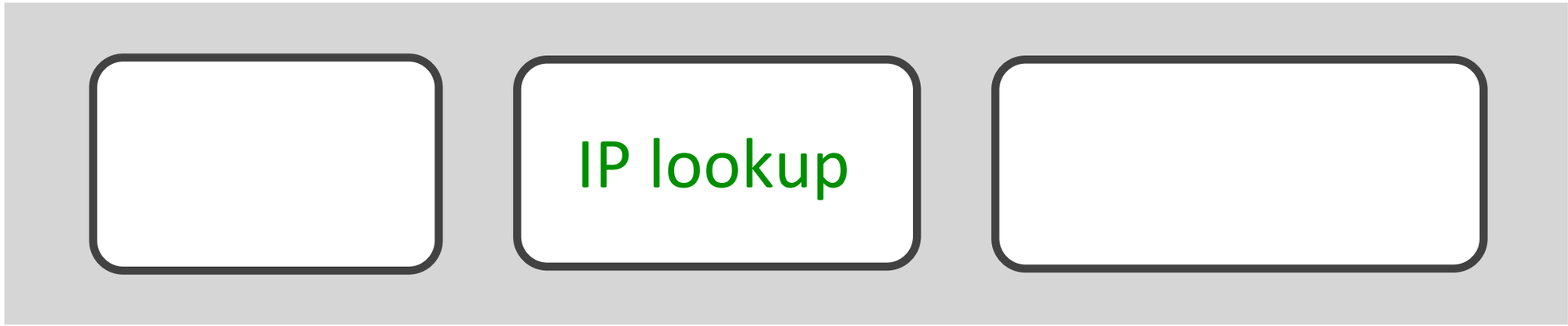
- Effect: compose at the iteration level
  - *can reduce #paths from* $\sim n^m$
  - *to* $\sim m\,n$

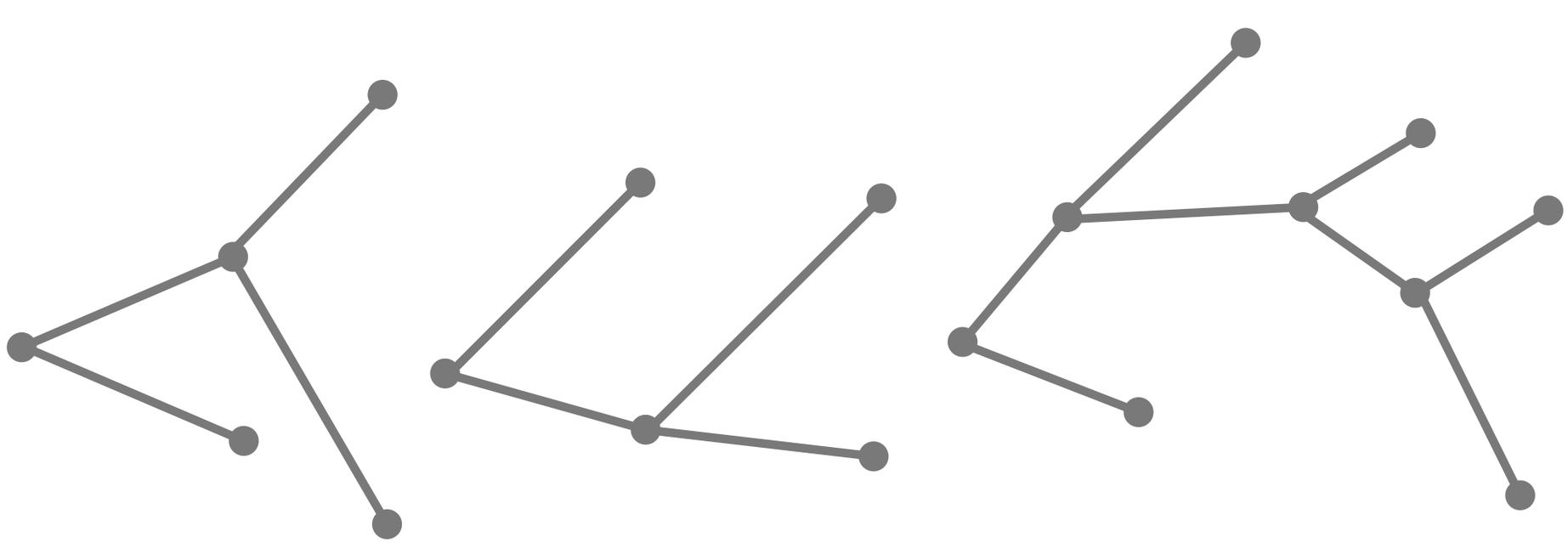
Outline

- Pipeline
- Loops
- Data structures
- Results

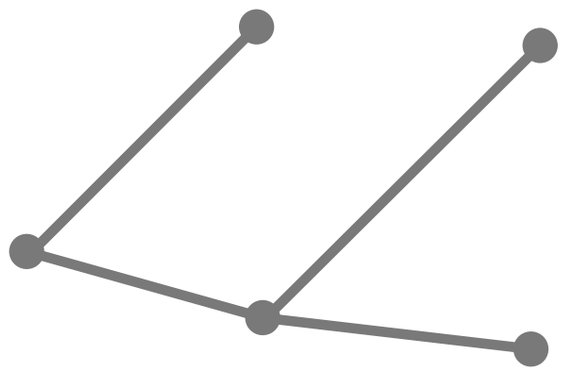





IP lookup

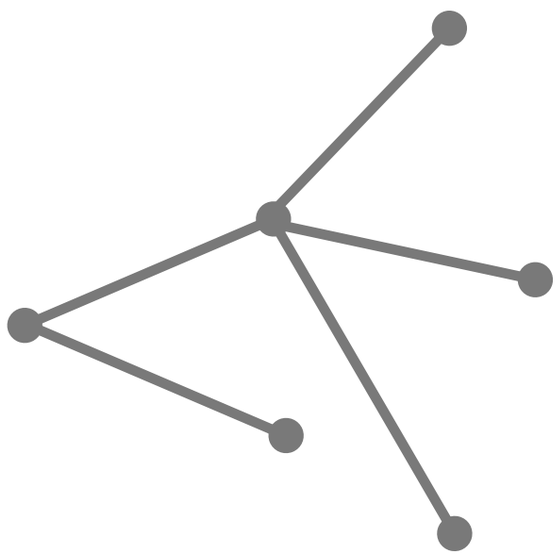


IP lookup

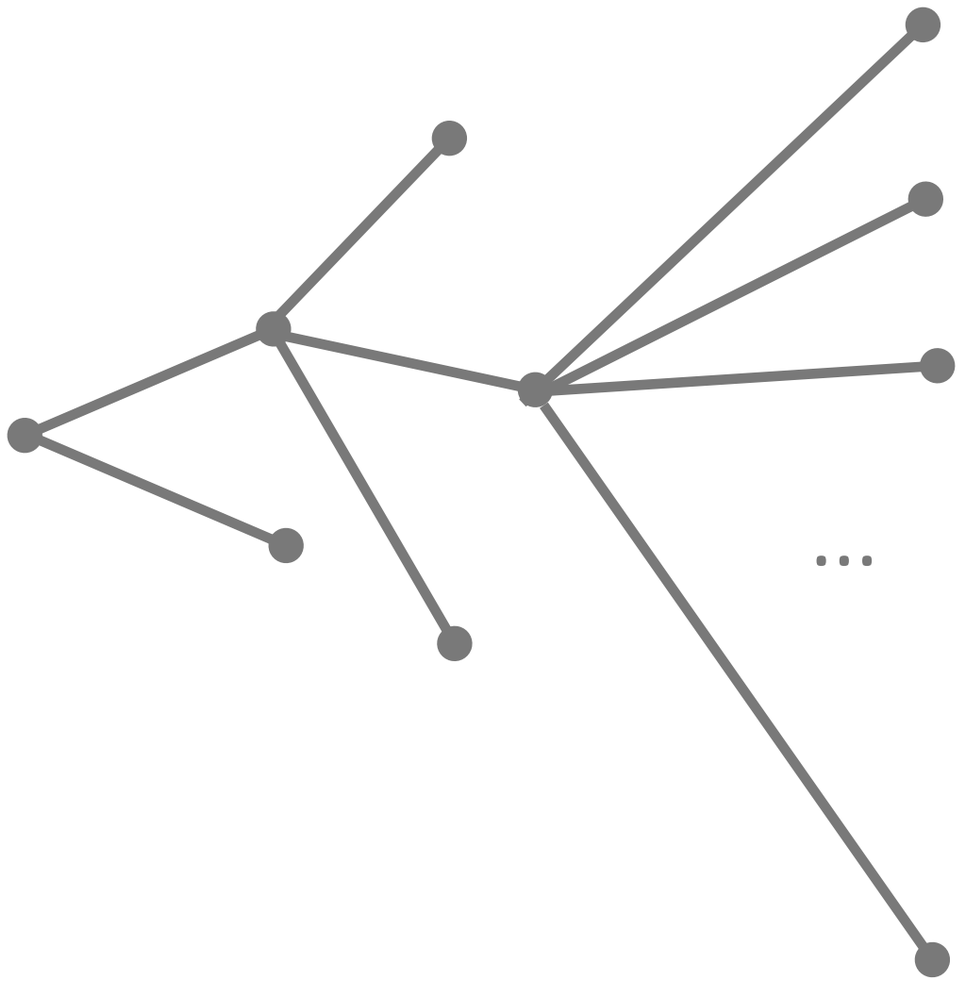


... output_port = table[dst_prefix] ...

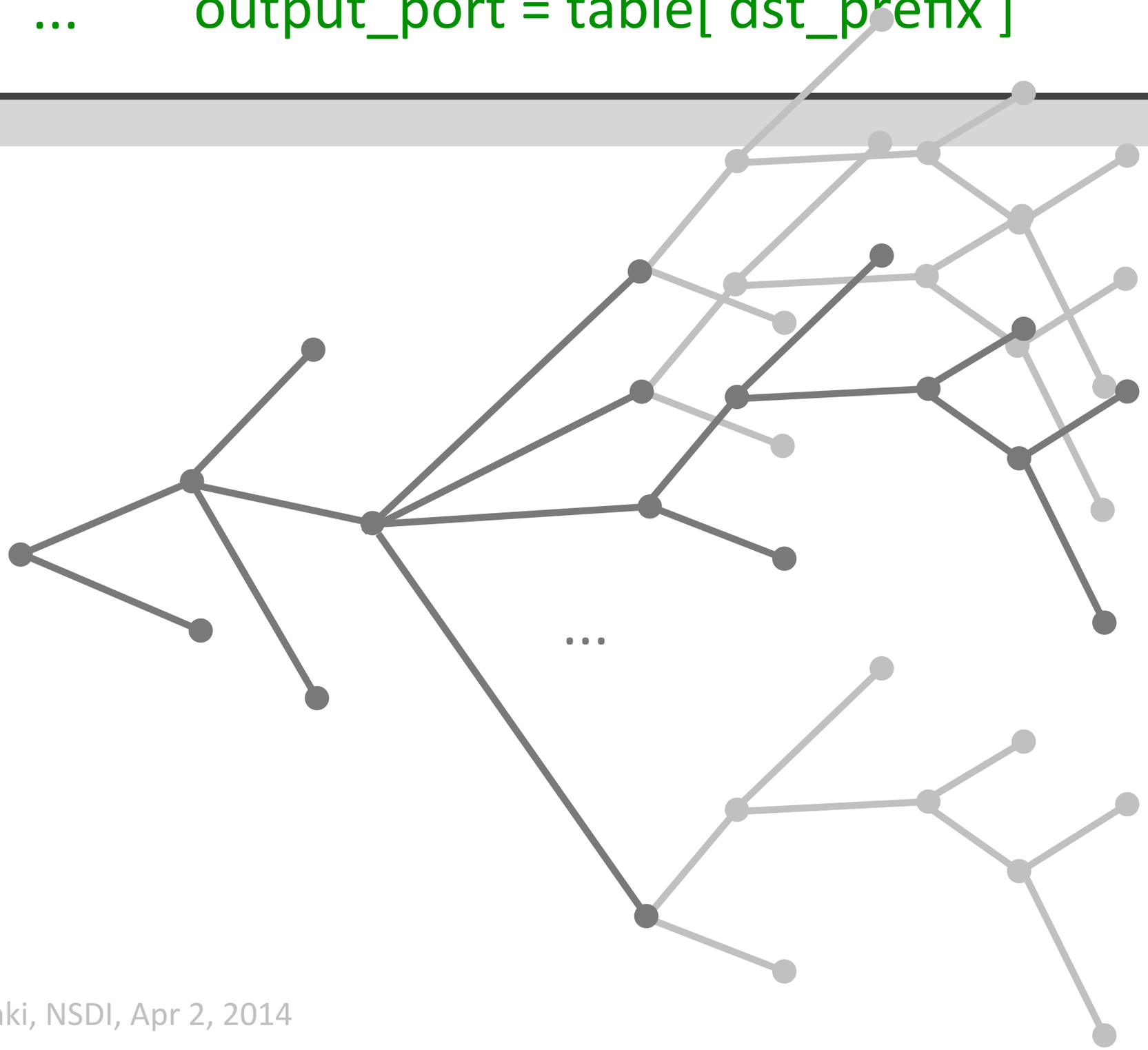
... output_port = table[dst_prefix] ...



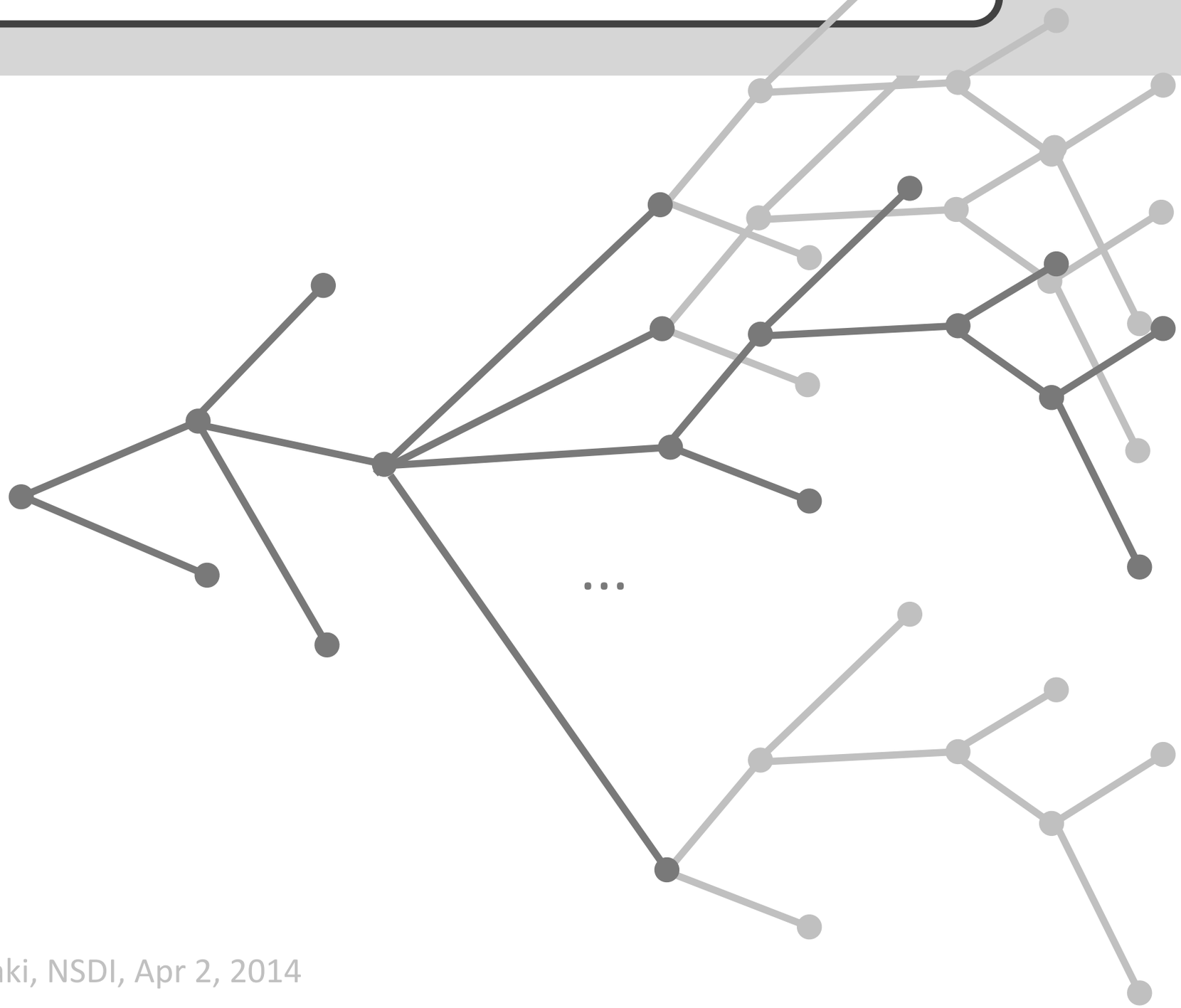
... output_port = table[dst_prefix] ...



... output_port = table[dst_prefix] ...

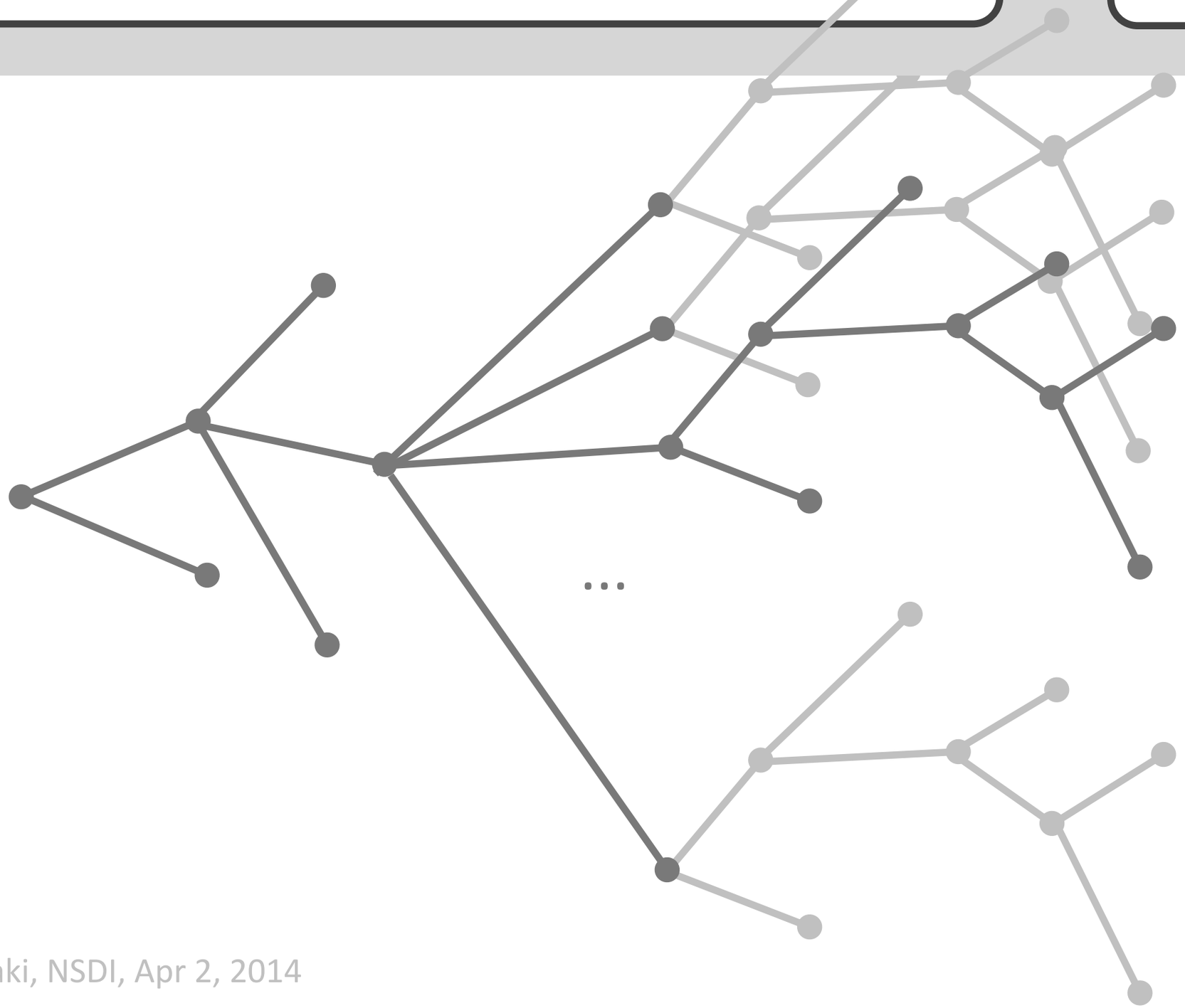


```
... output_port = table.read( dst prefix ) ...
```



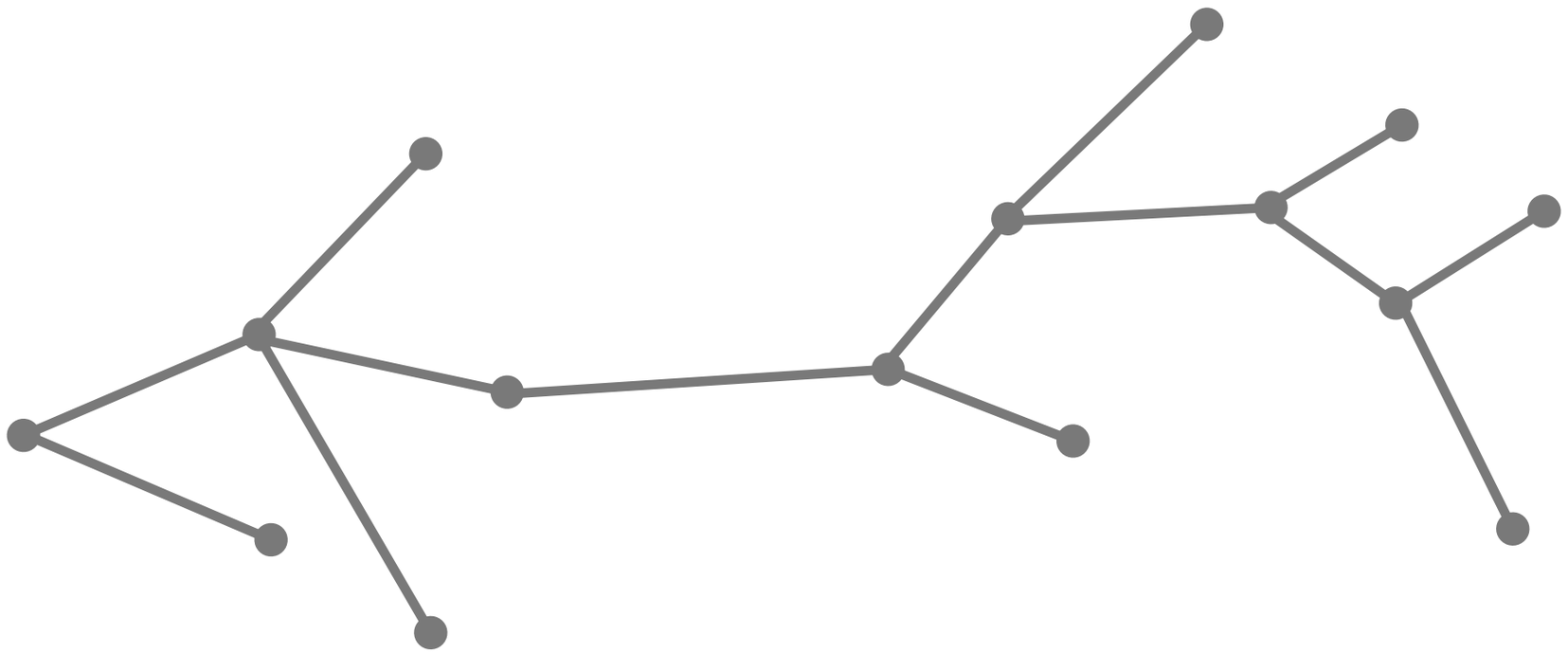
... output_port = table.read(dst prefix) ...

table impl



... output_port = table.read(dst prefix) ...

table impl



Data-access decomposition

- Rule: data-structure interface
  - *made explicit by the programmer*

- Effect: abstract data-structure implementation
  - *prevents data-structure size from contributing to path explosion*

Verified data structures

- Use pre-allocated arrays
  - *no dynamic memory (de)allocation*
  - *hash table, longest prefix match*

- Trade-off memory for “verifiability”
  - *at least as fast (array lookups)*
  - *but larger memory footprint (pre-allocation)*

Outline

- Pipeline
- Loops
- Data structures
- Results

Results

- Verified stateless & simple stateful pipelines
  - *IP router, NAT box, traffic monitor*
- Proved bounded execution
  - *no more than X instructions per packet*
  - *disparity between worst-case and common path*
- Proved crash-freedom
  - *no packet will cause the pipeline to abort*

```
/* IPFragmenter:: optcopy */

    for ( int i = 0; i < opts_len; ) {
       int opt = oin[i], optlen;
       if (opt == IPOPT_NOP)
          optlen = 1;
       else if (opt == IPOPT_EOL || i == opts_len - 1
       || i + (optlen = oin[i+1]) > opts_len)
          break;
       if (opt & 0x80) {
          //copy the option
          memcpy(...);
       }
       i += optlen;
  }
```

```
/* IPFragmenter:: optcopy */

    for ( int i = 0; i < opts_len; ) {
       int opt = oin[i], optlen;
       if (opt == IPOPT_NOP)
          optlen = 1;
       else if (opt == IPOPT_EOL || i == opts_len - 1
       || i + (optlen = oin[i+1]) > opts_len)
          break;
       if (opt & 0x80) {
          //copy the option
          memcpy(...);
       }
       i += optlen;
  }
```

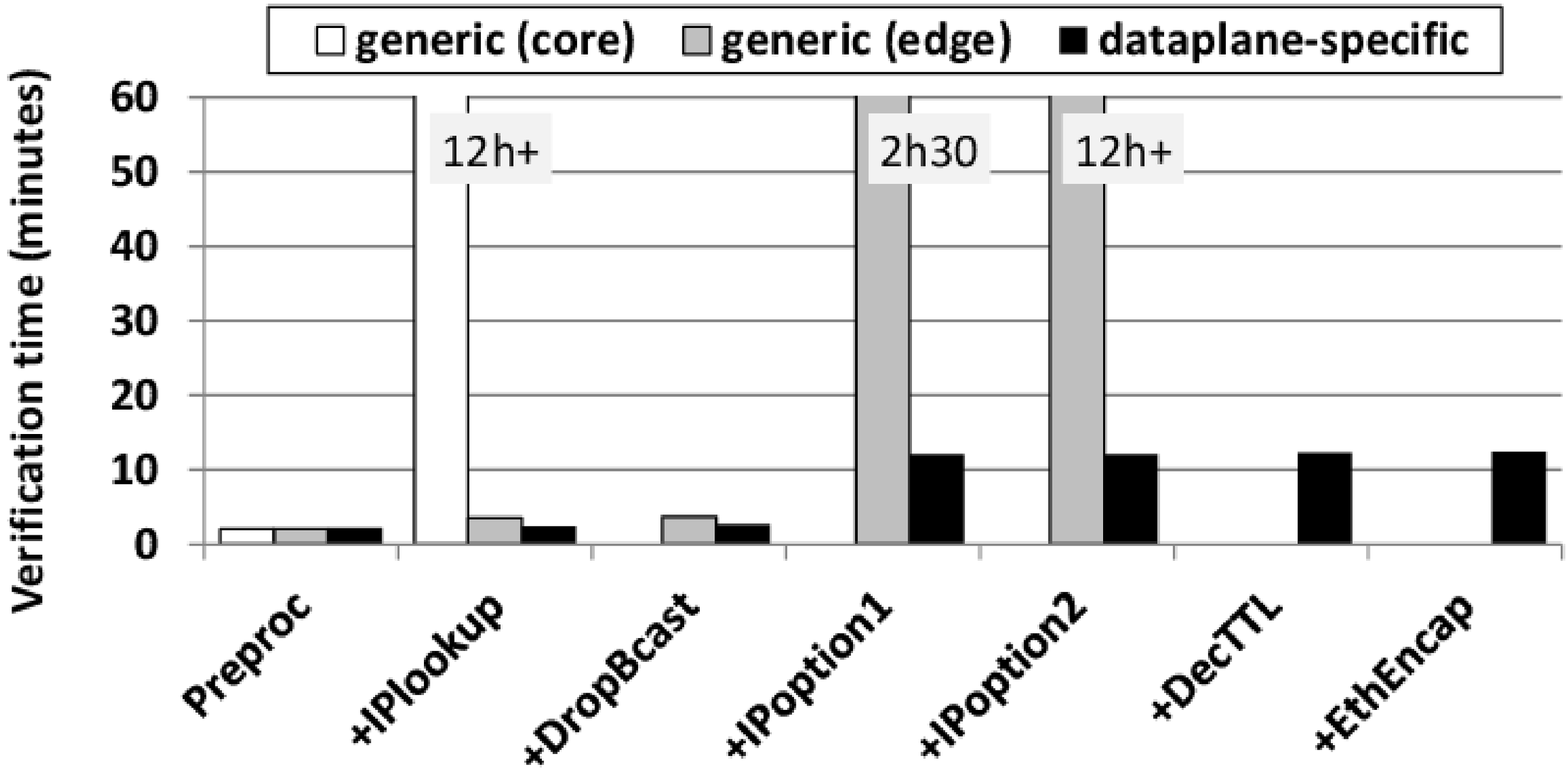
```
/* IPFragmenter:: optcopy */

    for ( int i = 0; i < opts_len; ) {
        int opt = oin[i], optlen;
        if (opt == IPOPT_NOP)
            optlen = 1;
        else if (opt == IPOPT_EOL || i == opts_len - 1
        || i + (optlen = oin[i+1]) > opts_len)
            break;
        if (opt & 0x80) {
            //copy the option
            memcpy(...);
        }
        i += optlen;
    }
```

```
/* IPFragmenter:: optcopy */

    for ( int i = 0; i < opts_len; ) {
        int opt = oin[i], optlen;
        if (opt == IPOPT_NOP)
            optlen = 1;
        else if (opt == IPOPT_EOL || i == opts_len - 1
        || i + (optlen = oin[i+1]) > opts_len)
            break;
        if (opt & 0x80) {
            //copy the option
            memcpy(...);
        }
        i += optlen;
    }
```

Verification time for Click pipelines



Homage

- Active networks
  - *Tennenhouse & Wetherall, CCR 1996*
- S2E software analyzer
  - *Chipounov et al., ASPLOS 2011*
- Compositional analysis
  - *Godefroid, POPL 2007*
- Click programming framework
  - *Kohler, PhD thesis, 2000*

Conclusion

- Dataplane-specific verification
  - *symbolic execution + composition*
  - *pipeline structure, limited loops, pre-allocated key/value stores*

- Enables dataplane verification in useful time
  - *complete and sound analysis*
  - *of stateless and 2 simple stateful pipelines*