Distributed traces modeling and critical path analysis

Progress Report Meeting
December 6th 2012

Francis Giraldeau
francis.giraldeau@polymtl.ca

Under the direction of Michel Dagenais
DORSAL Lab, École Polytechnique de Montréal

Plan

- Research objectives
- Execution graph recovery
- Critical path computation
- Future work

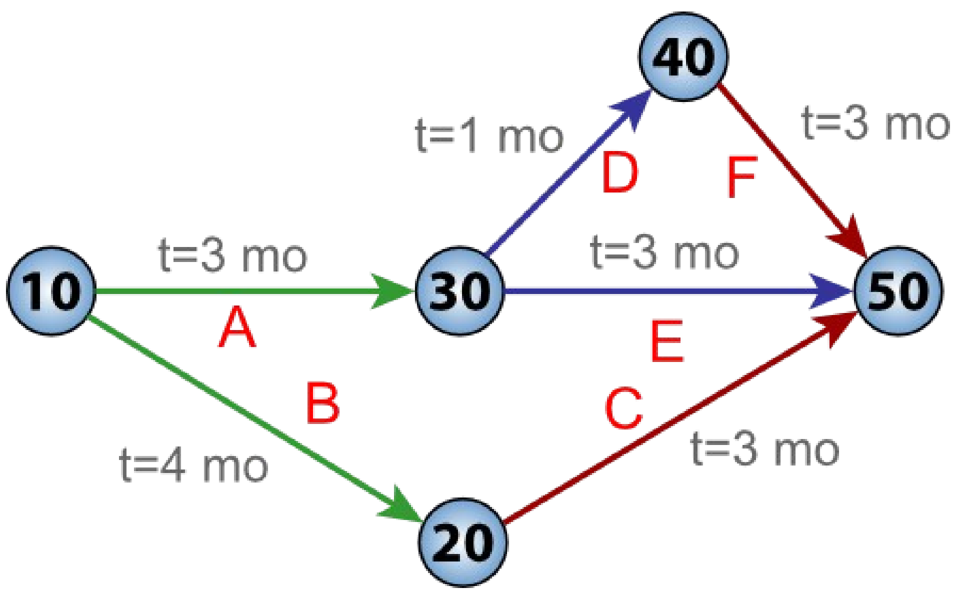
Objectives

General objective

“ *Provide trace analysis tools to understand the overall performance of a distributed application.* ”

Critical Path Method

- Used in project management (PERT, Gantt)
- Directed Acyclic Graph (DAG) of activities
- The critical path is the longest path in the graph



Drawing source: wikipedia.org

Detailed objectives

1. Develop instrumentation and semantic to extract execution graph from kernel trace
2. Extract distributed execution graph online
3. Determine the critical path
4. Calculate resources usage of execution

Black box approach required

Assumption: linux kernel is used

Research questions

“ *Is it possible to extract the critical path of a distributed application from a kernel trace?* ”

“ *If so, what is the most efficient and reliable way to perform this analysis online?* ”

Literature review

Observation of distributed systems

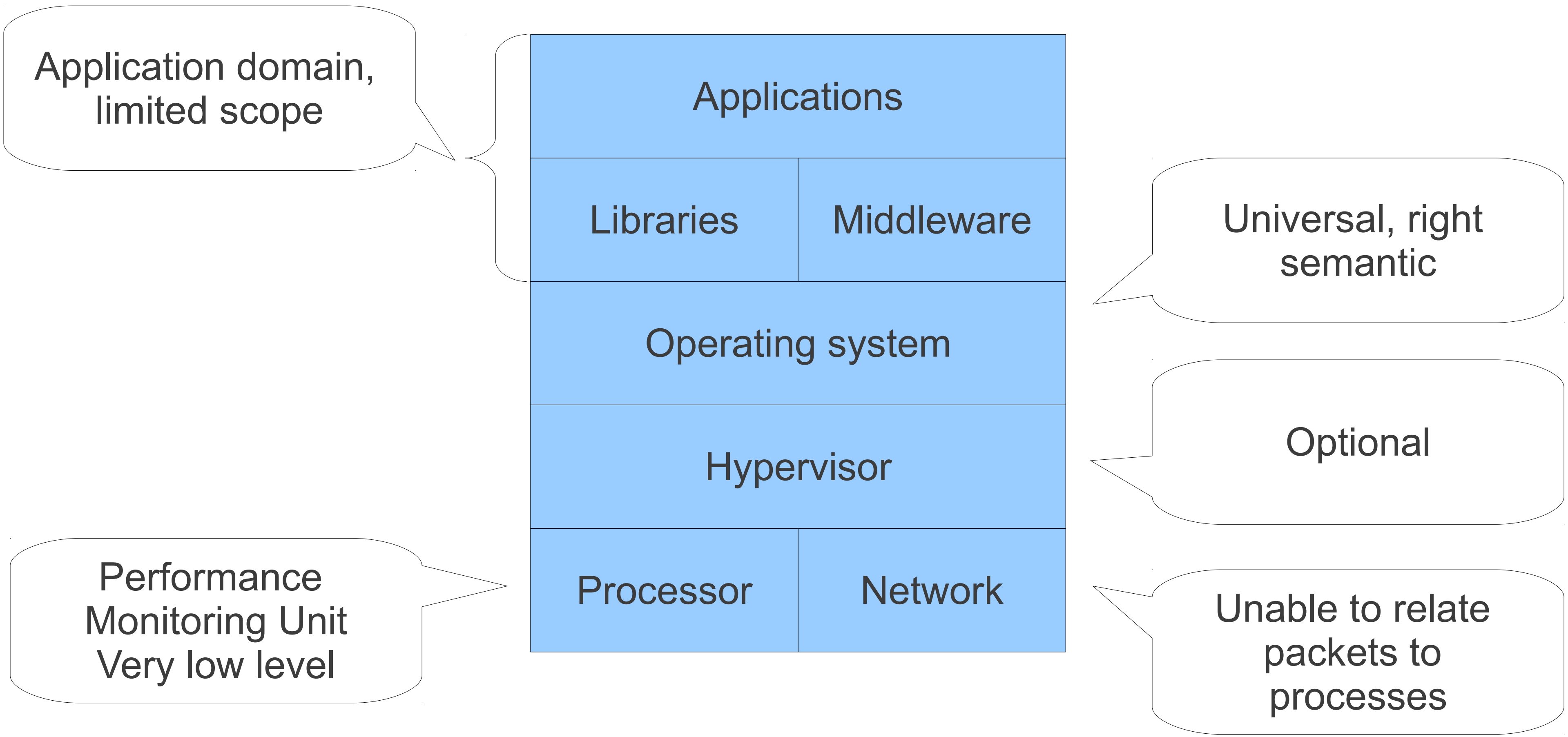
Precise

- Systematic event processing
- Accurate measure
- Scalability issue
- Subject to event loss

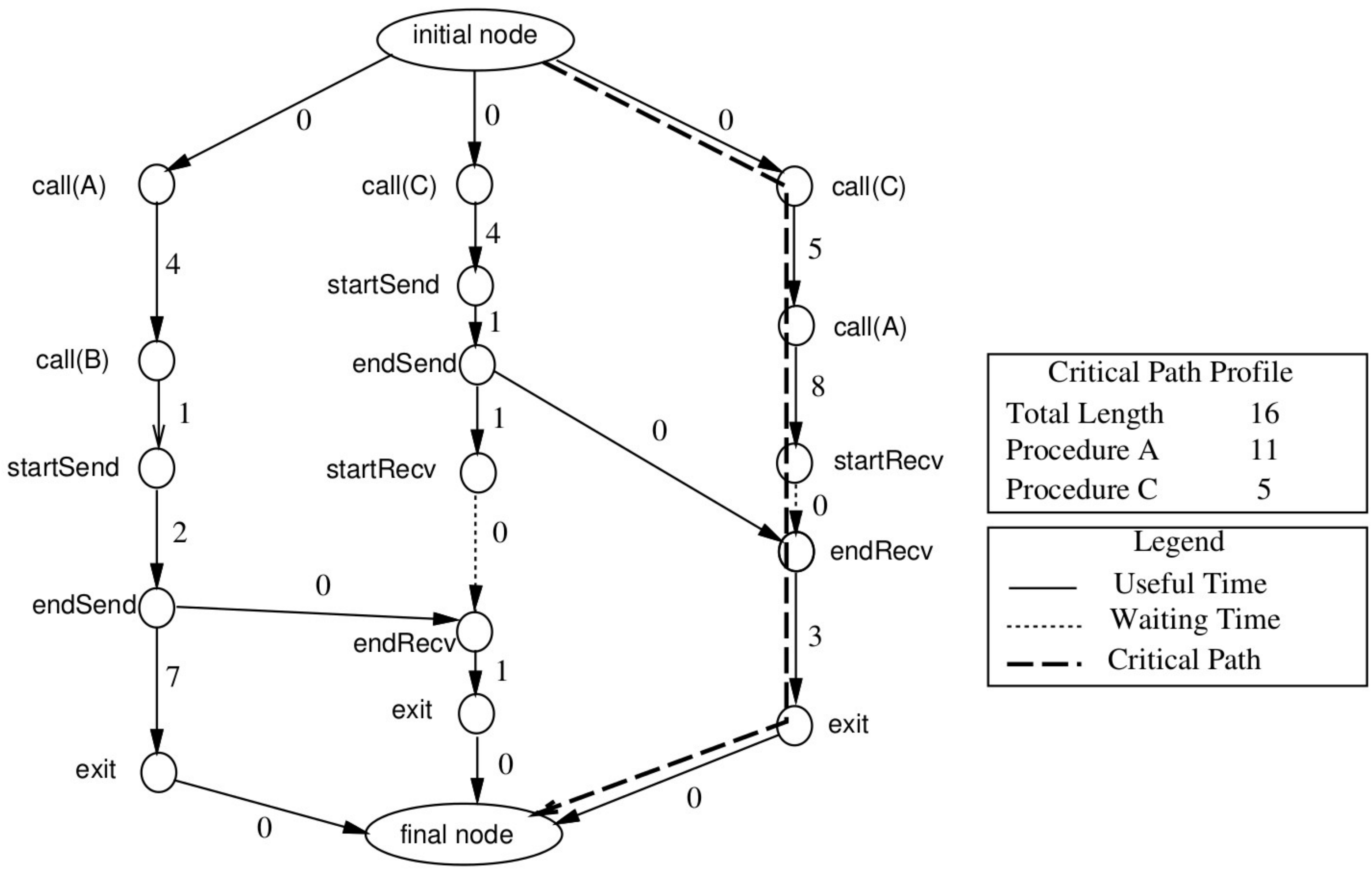
Statistical

- Event sampling
- Scalable
- Allow false positive

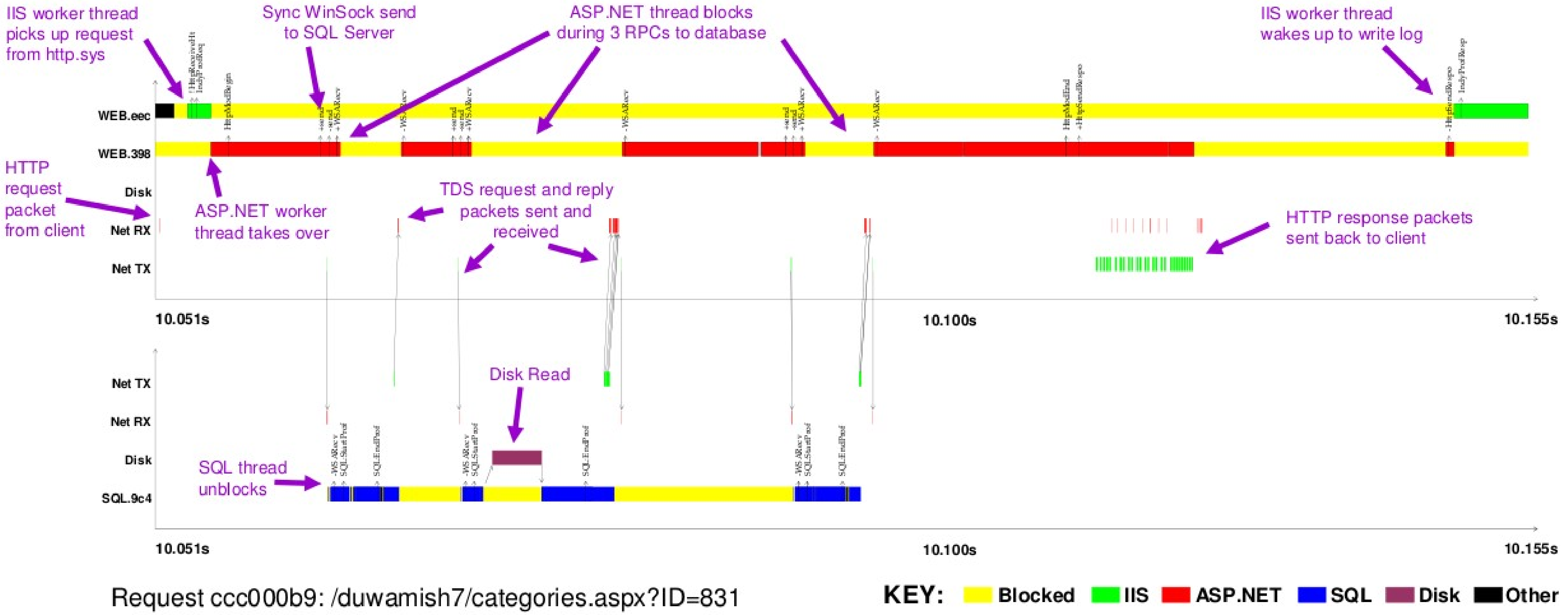
Instrumentation level



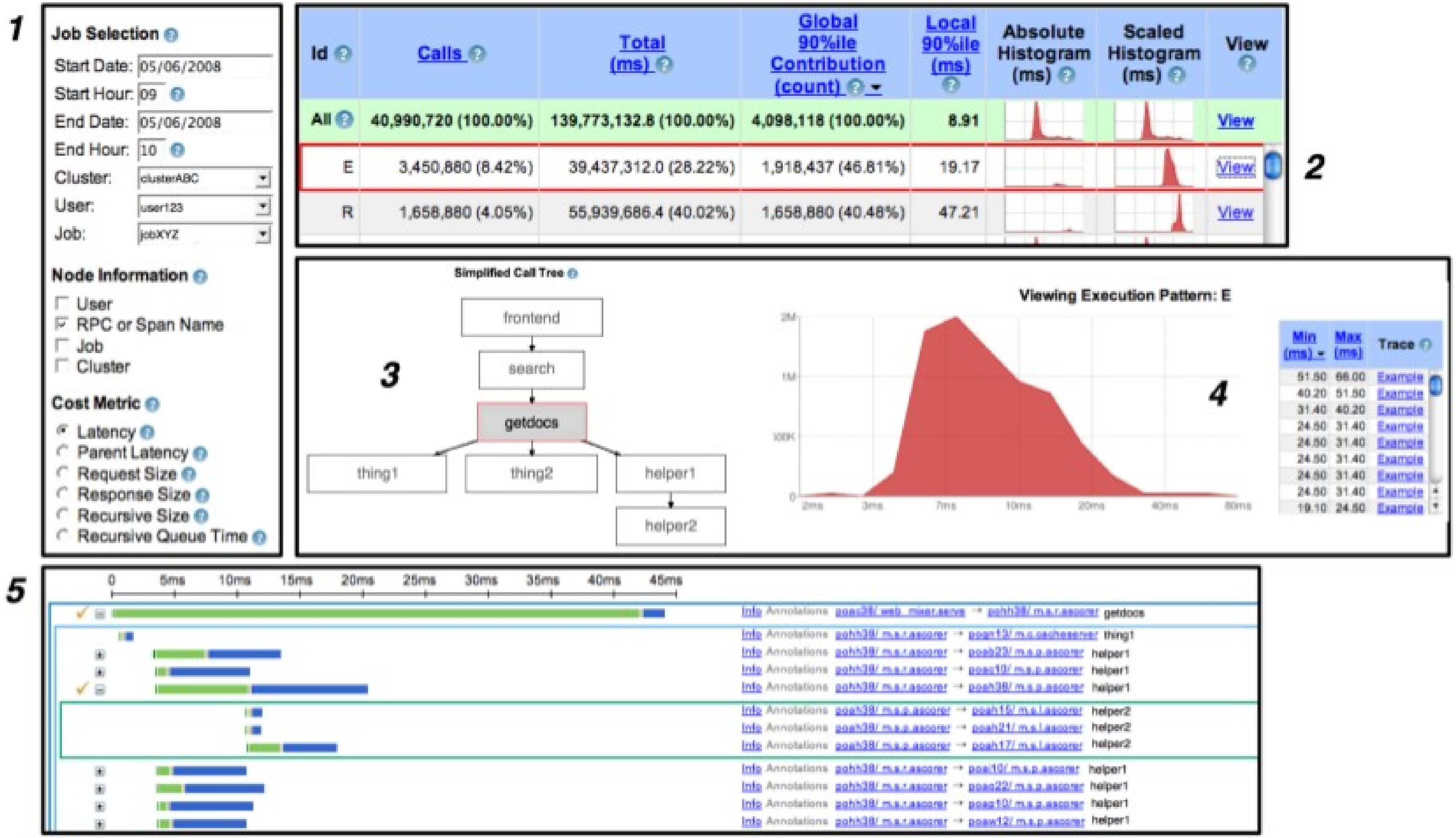
CPU critical path [42]



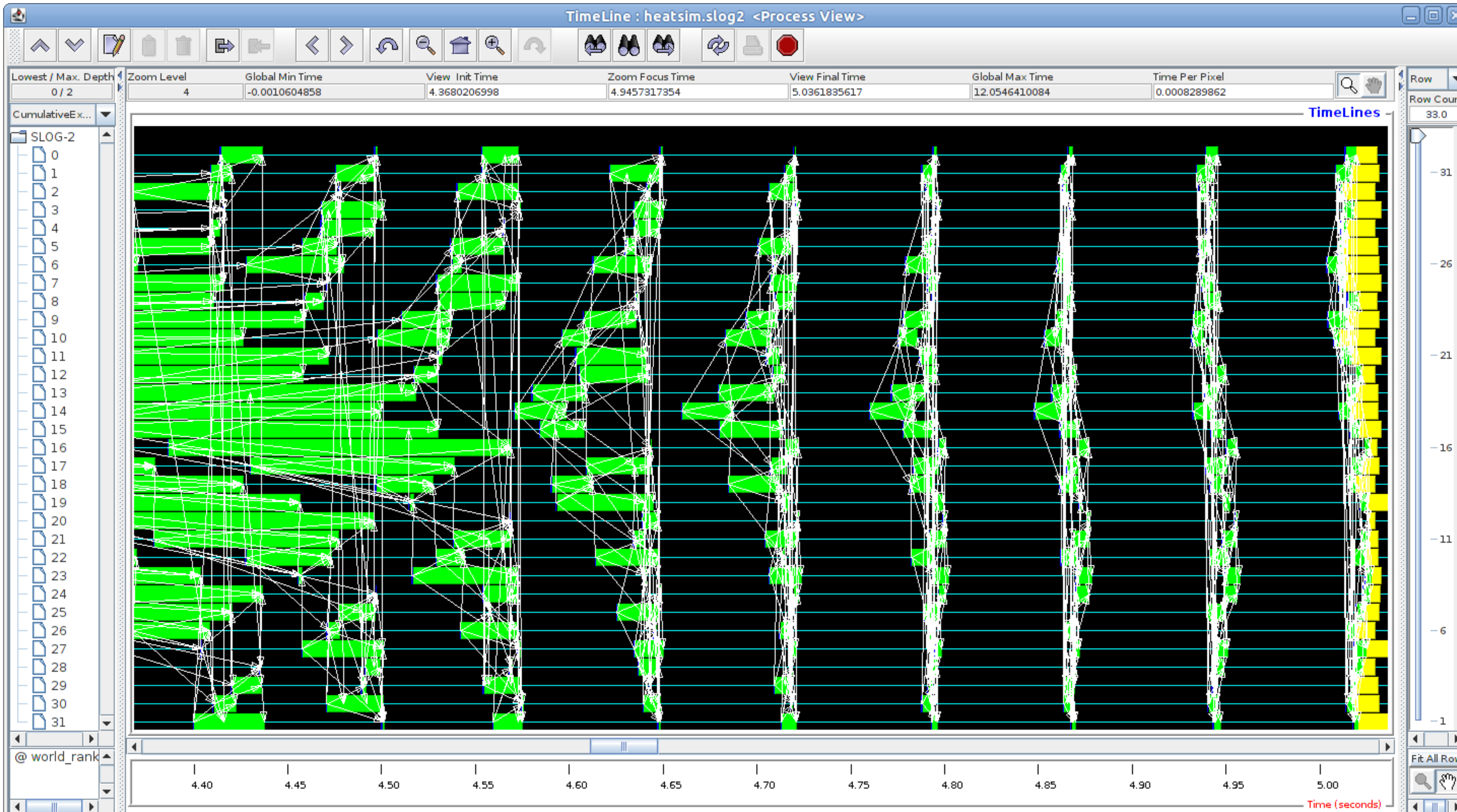
Microsoft's Magpie [30]



Google's Dapper [21]



MPI Jumpshot [22]



Summary

- Focused on network
  - Unable to recover process relationship
- Requires software adaptation
  - Incompatible with black box approach
- Limited scope
  - Focused on three-tiers infrastructure
  - Specific to application, library or middleware

Methodology

Methodology

- Design small programs with known behavior
  - Project workload-kit to generate a standard traceset
- Run programs while kernel tracing is enabled
- Analyze trace manually
  - Recover system state
  - Correlate events among objects
  - Validate assumptions
  - Highlight limitations
- Modify kernel instrumentation (as modules)

Workload-kit

- Calibrated CPU hog
- Burst I/O sync/async
- Synchronization (deadlock, pipeline, imbalance)
- TCP/UDP network transmission

Wait analysis

- Types of waiting
  - Preempted (ex: quantum exhausted)
  - Interrupted (ex: IRQ)
  - Blocking (ex: cold read on disk)
- Passive wait mechanism
- Occurs always in system calls
- Different wait state
- Wait source is on the critical path

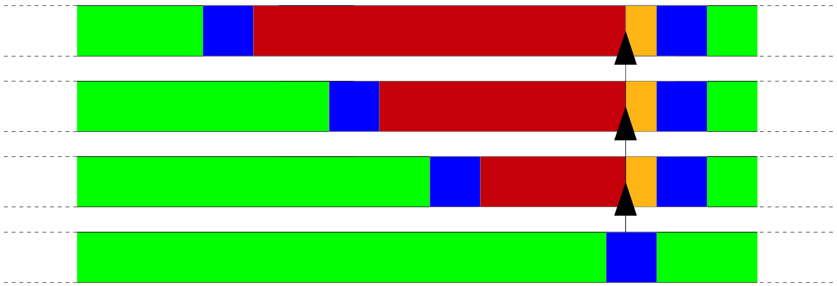
Main system calls

System call	Control flow effect	Wake-up source
nanosleep	No change	Timer
read	Change to device	Softirq
write (sync)	Change to device	Softirq
waitpid	Change to local task	task
futex	Change to local task	task
recv	Change to network and remote task	Softirq

Bypassing system calls

OMP_WAIT_POLICY=ACTIVE

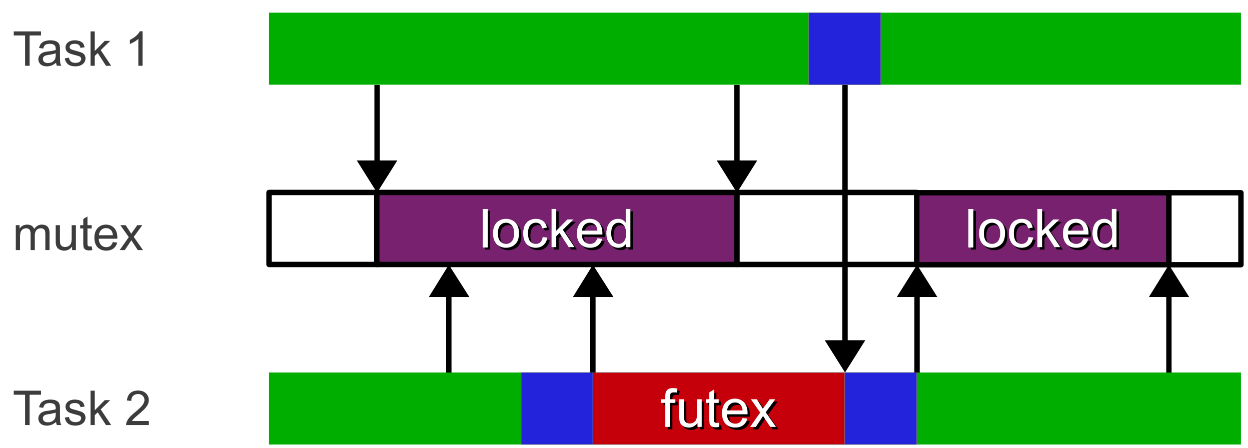
OMP_WAIT_POLICY=PASSIVE



- Spin locks are usually short duration
- Analysis relies on system calls

Futex synchronization

- Stays in userspace if no contention
- Lock held, want lock: FUTEX_WAIT
- On unlock, wake pending: FUTEX_WAKE



Shared memory

- Communication by shared memory is not visible from kernel space by default
- Could be instrumented with traps (page fault) but very costly

Asynchronous system calls

- Increase parallelism
- Still need some synchronization
- Do not affect critical path recovery

Userspace thread

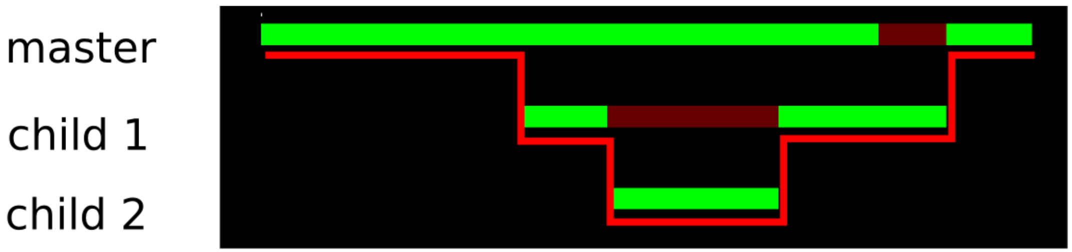
- Appears as a single process
- The system level execution can be recovered
- Require threading library instrumentation

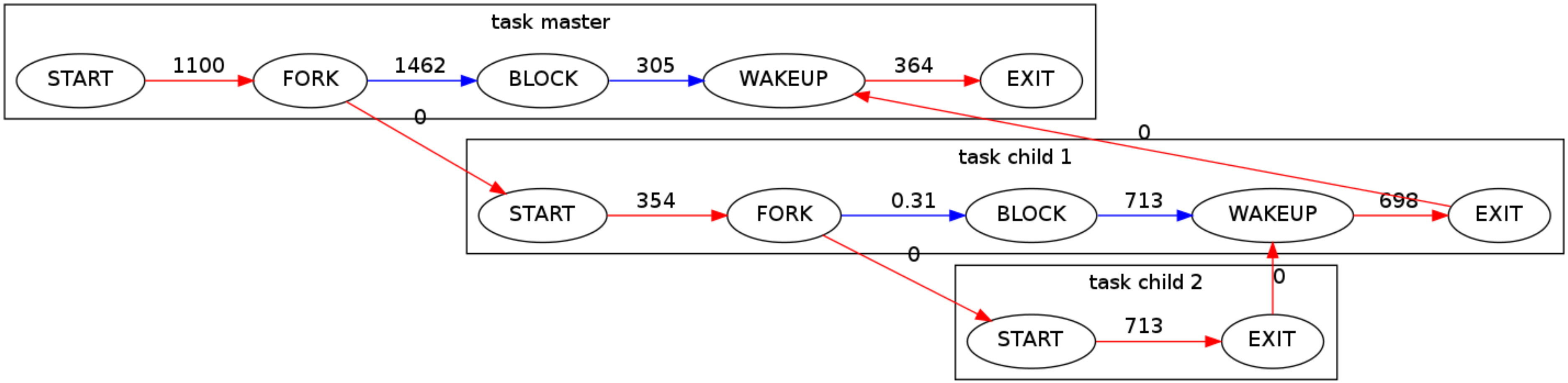
Execution graph recovery

Execution graph semantic

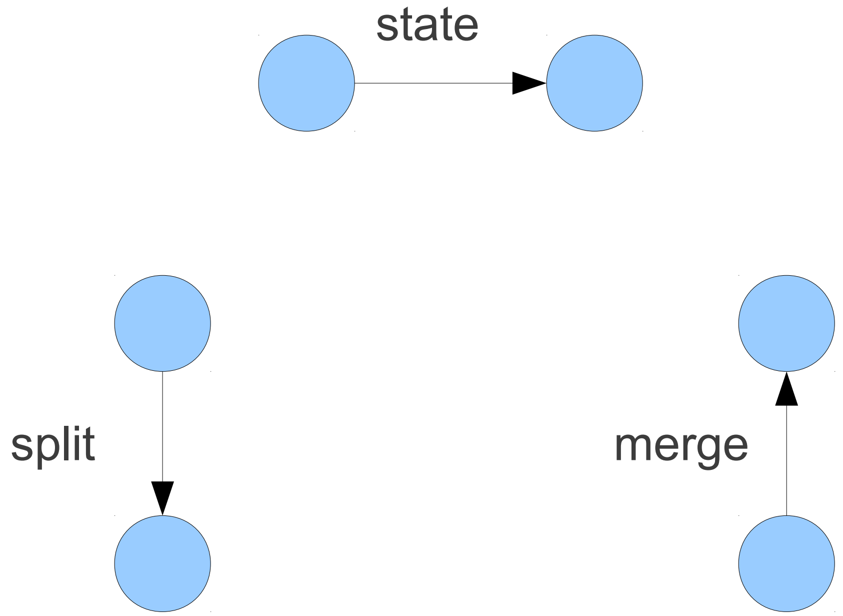
- Directed acyclic graph
- Actor: system object
  - Task, mutex, fd, sock, etc.
- Vertex: key execution events
  - Fork, wakeup, read, write, etc.
- Per actor edge: actor state
  - Wait, busy, running, etc.
- Cross actor edge: links between objects
  - Split or merge

Fork/waitpid example

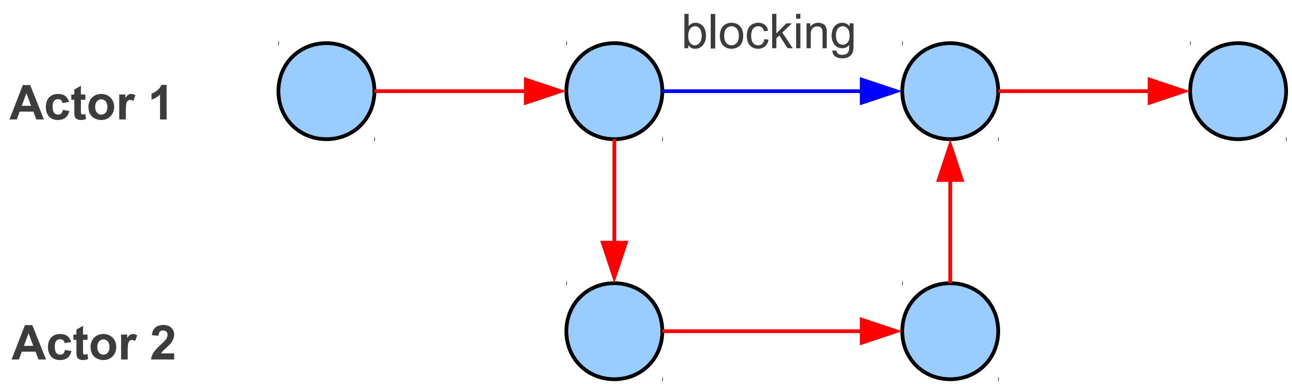




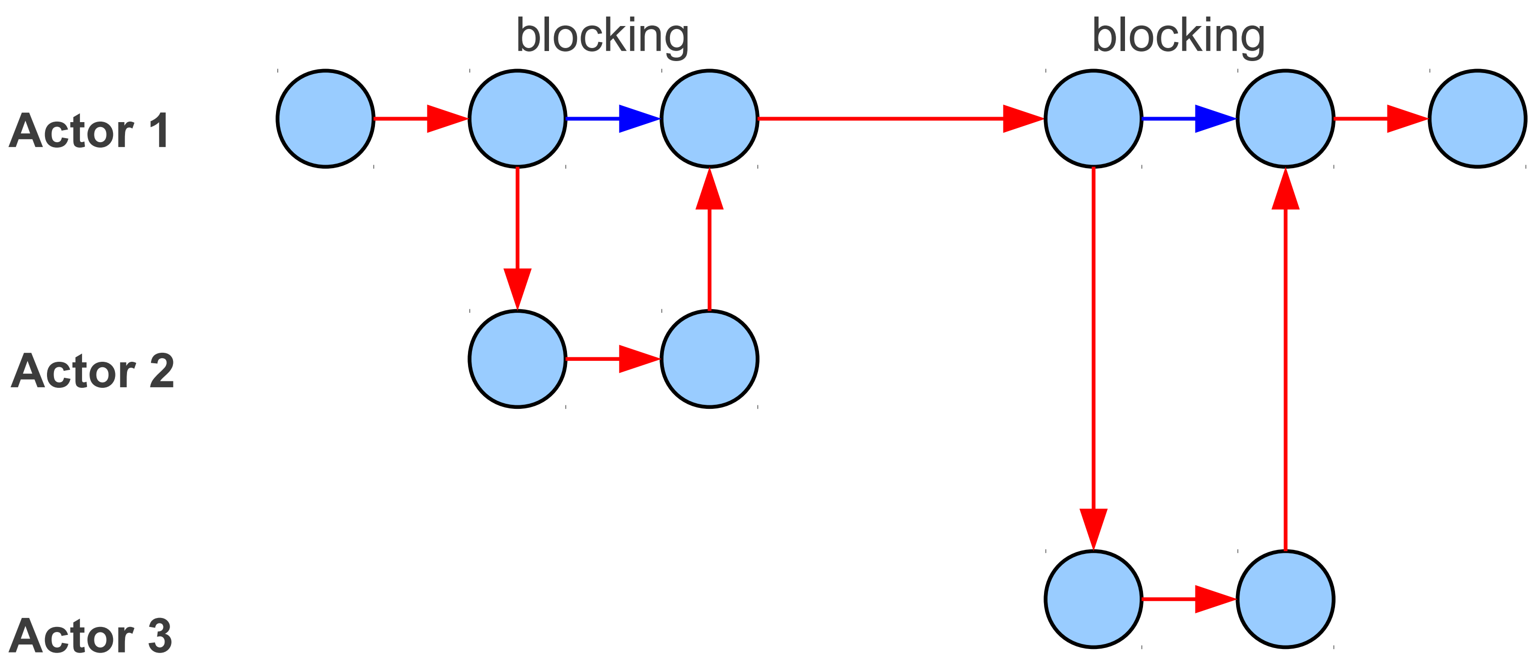
Basic graphs



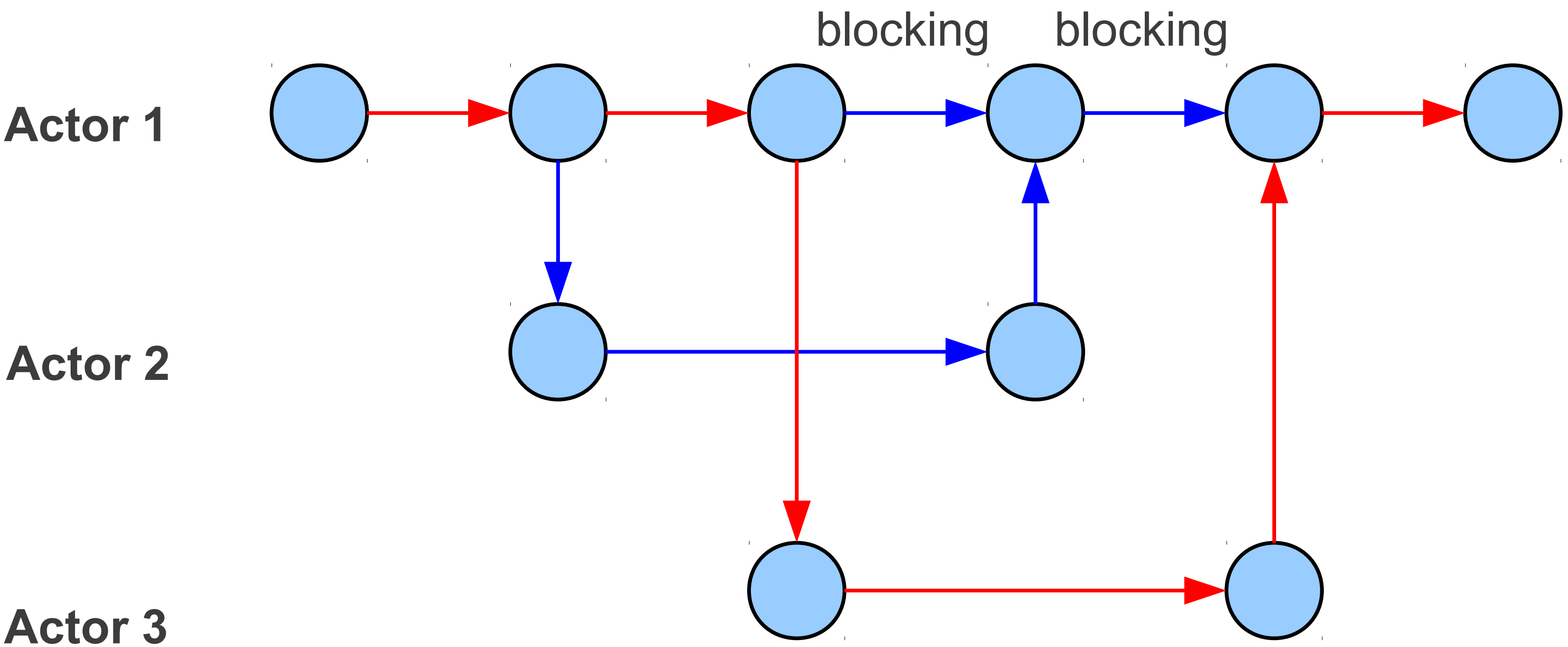
Basic graphs



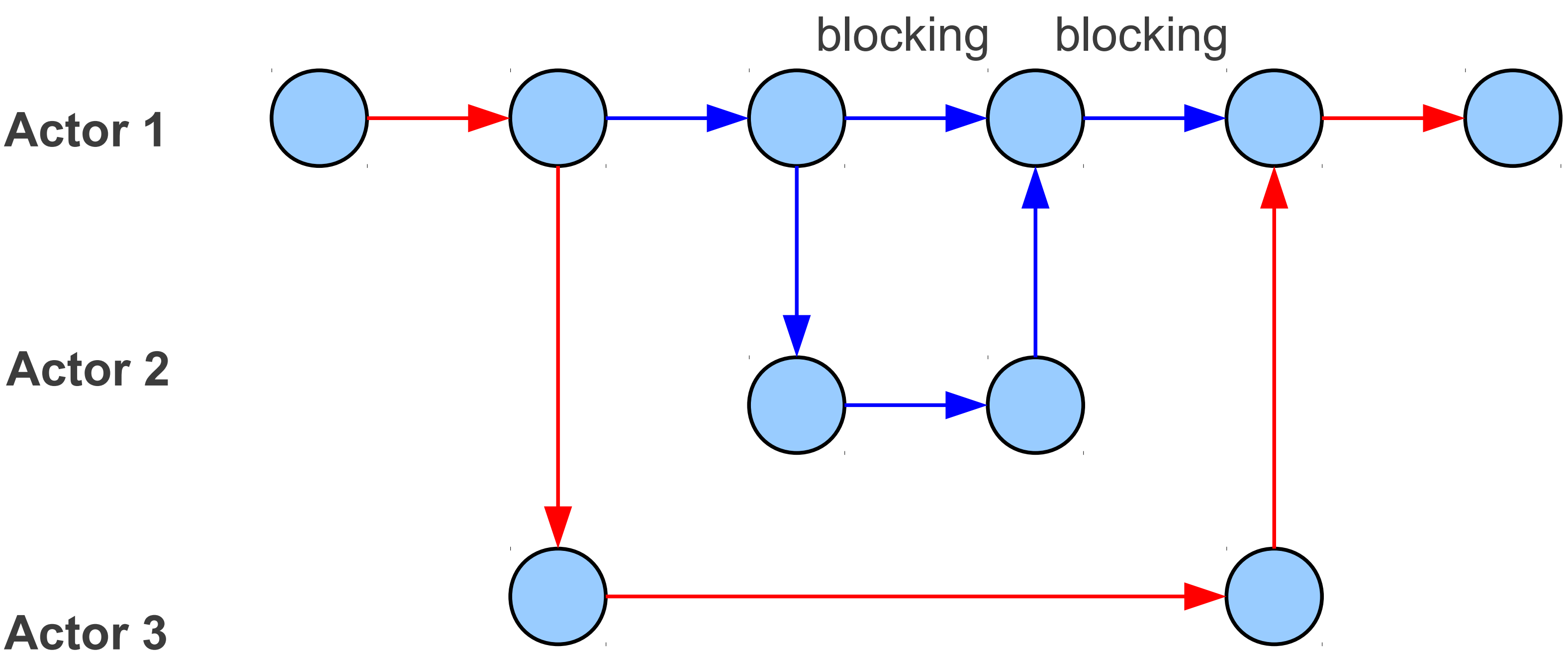
Basic graph concatenation



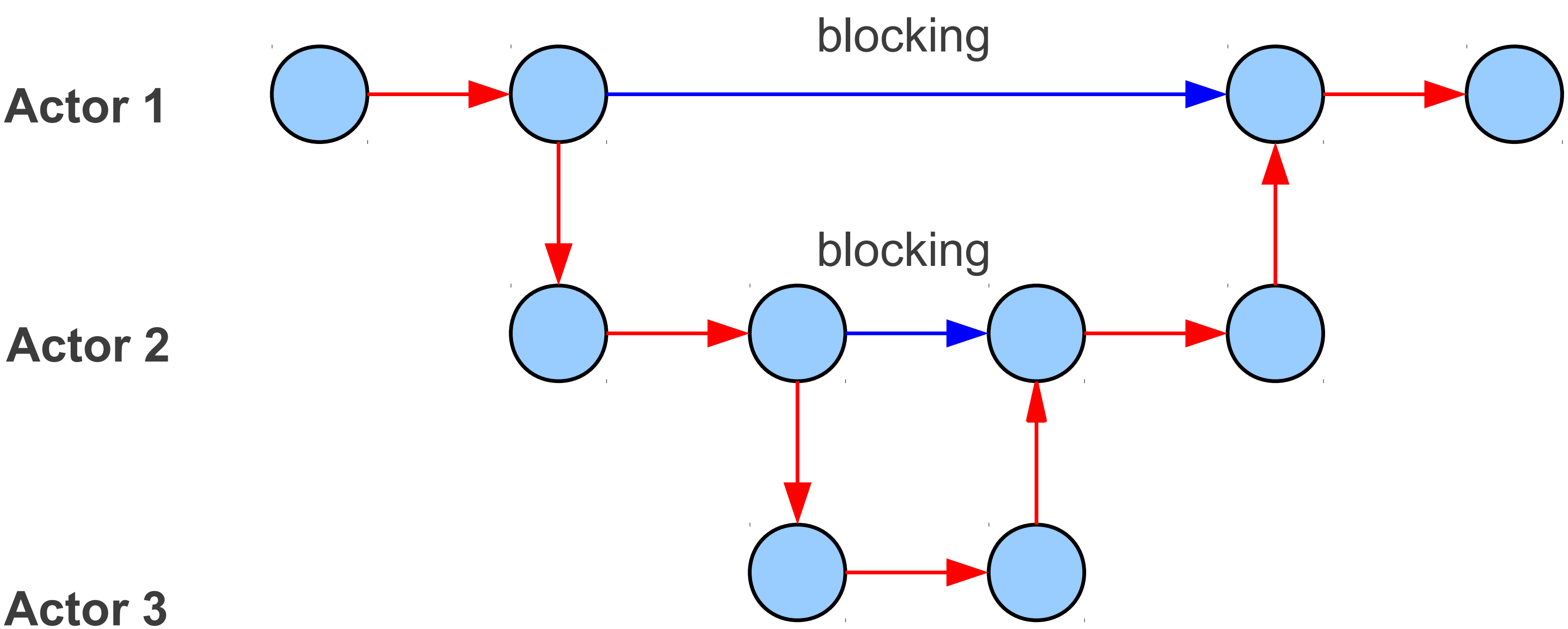
Basic graph interleaved



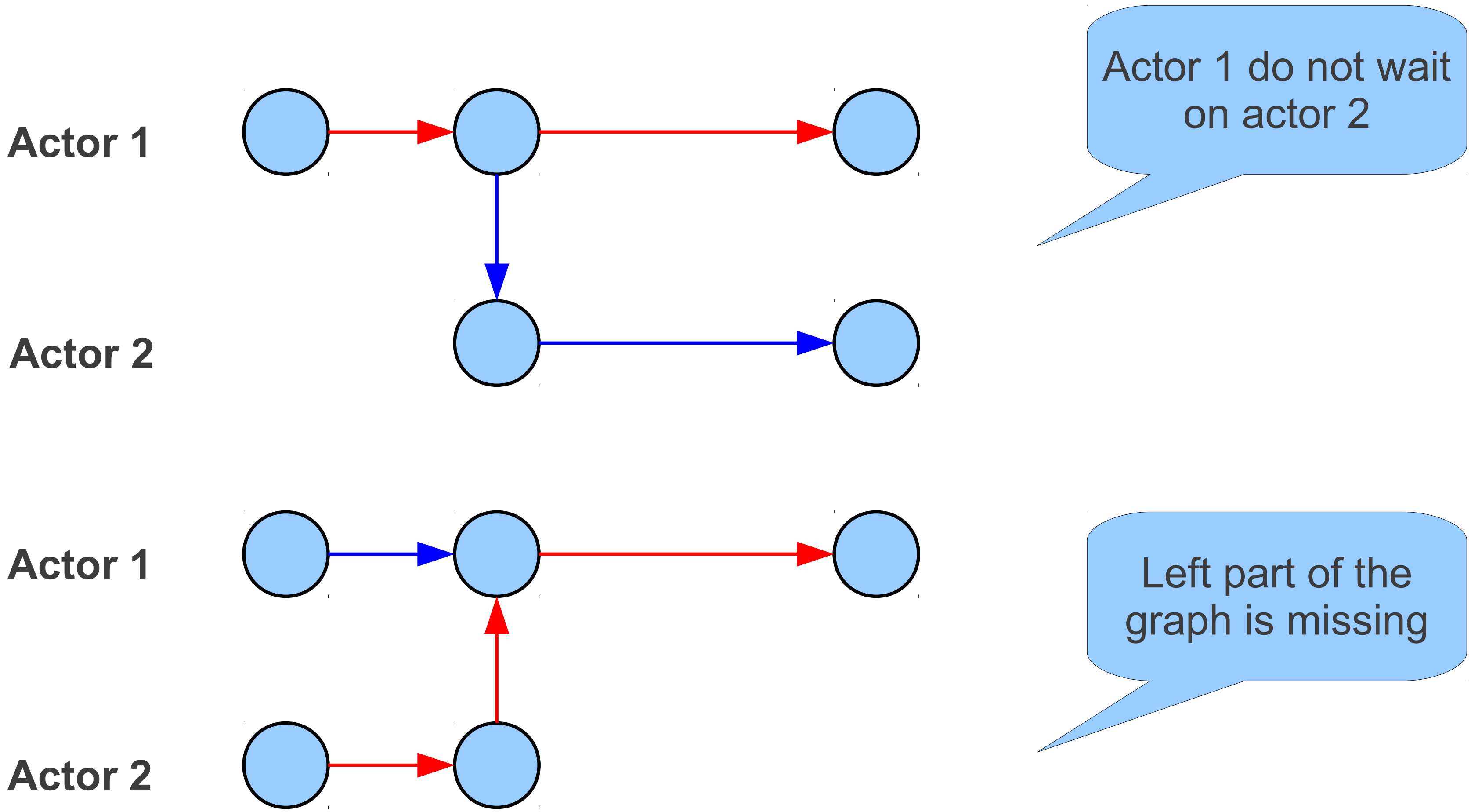
Basic graph embed



Basic graph nested



Basic graph opened



Critical Path Analysis

Critical path computation

- Backward algorithm
  - Simplest method
  - Requires full graph in memory
  - Not suitable for on-line analysis
- Forward algorithm
  - Breadth first search with O(n) complexity
  - Closest-first traversal
  - Incremental path pruning
  - Suitable for on-line analysis

Critical path algorithm

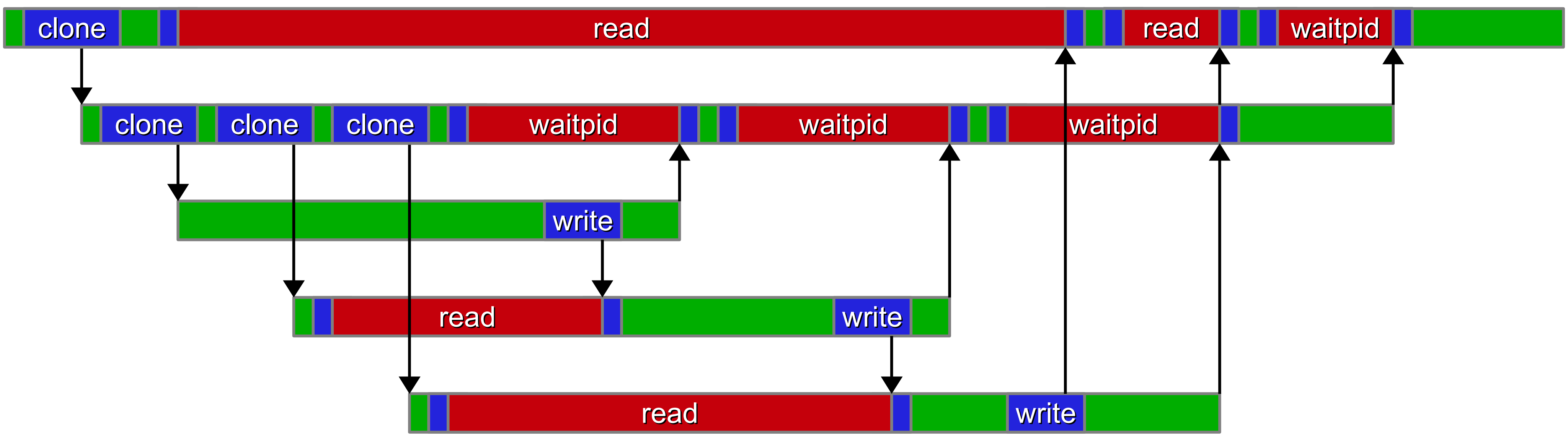
- Closest-first breadth first search iterator
- Annotate each visited edge as candidate
- If blocking (except self-wait) encountered, annotate edges backward as non-critical path until reaching a node that has two candidate edges node with reached
- The result is annotated critical path.
- The critical path may not be unique.

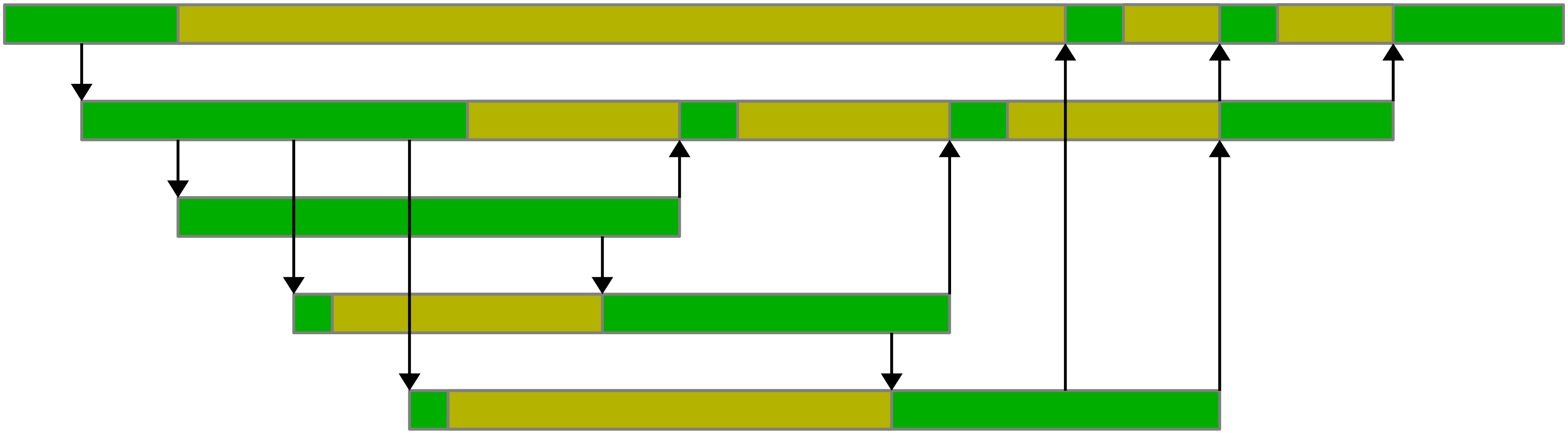
Example of critical path computation

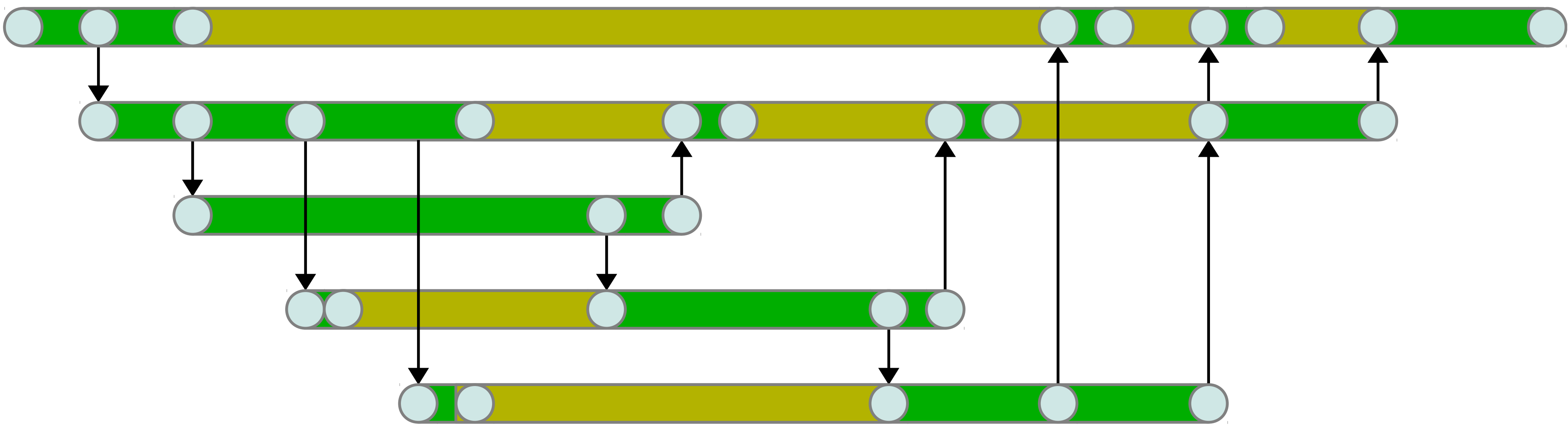
```
#!/bin/sh
V=$(ls | tail | grep)
```

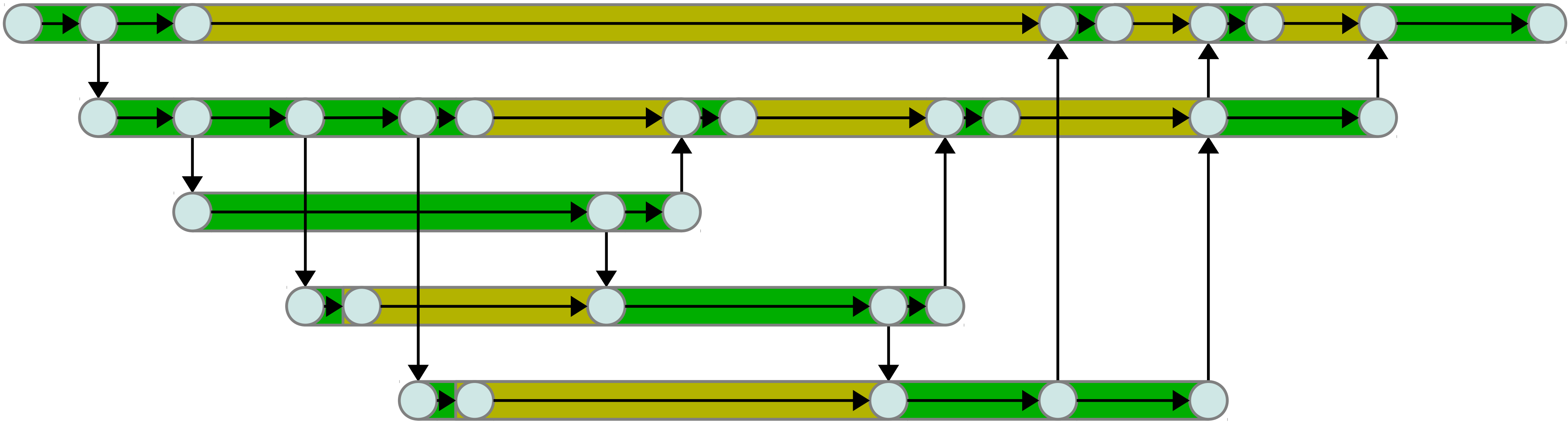
5 processes are involved:

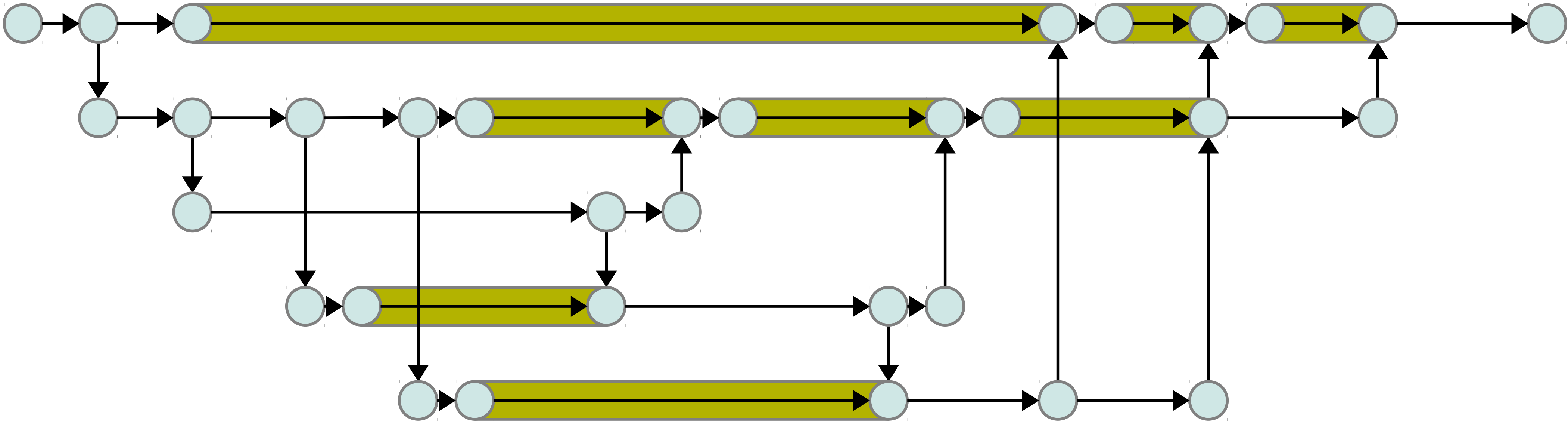
- 1 sh
- 2 sh
- 3 ls
- 4 tail
- 5 grep

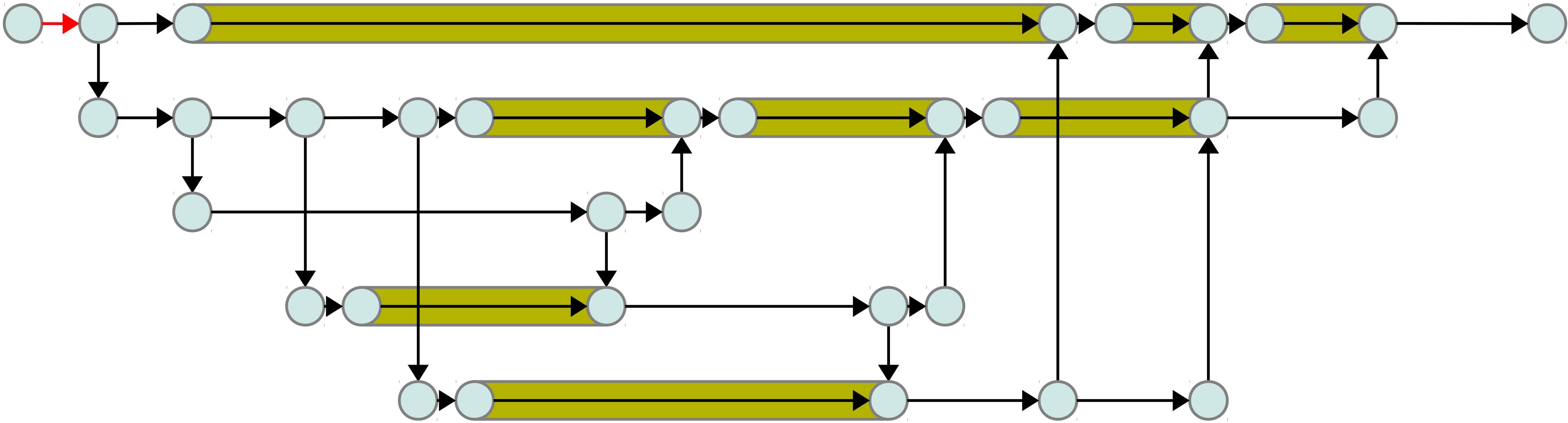


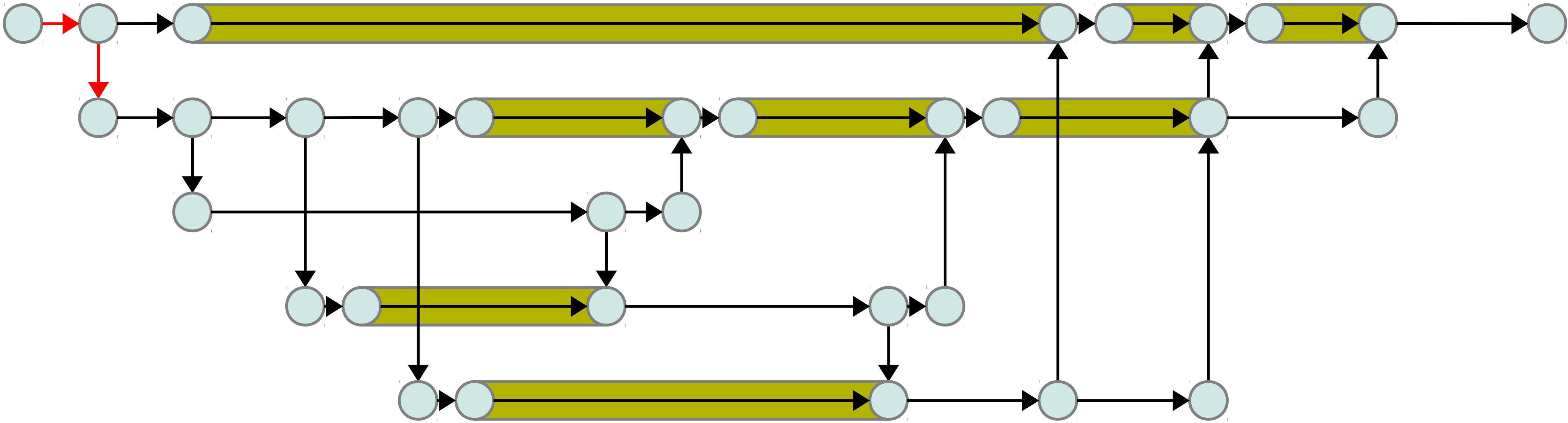


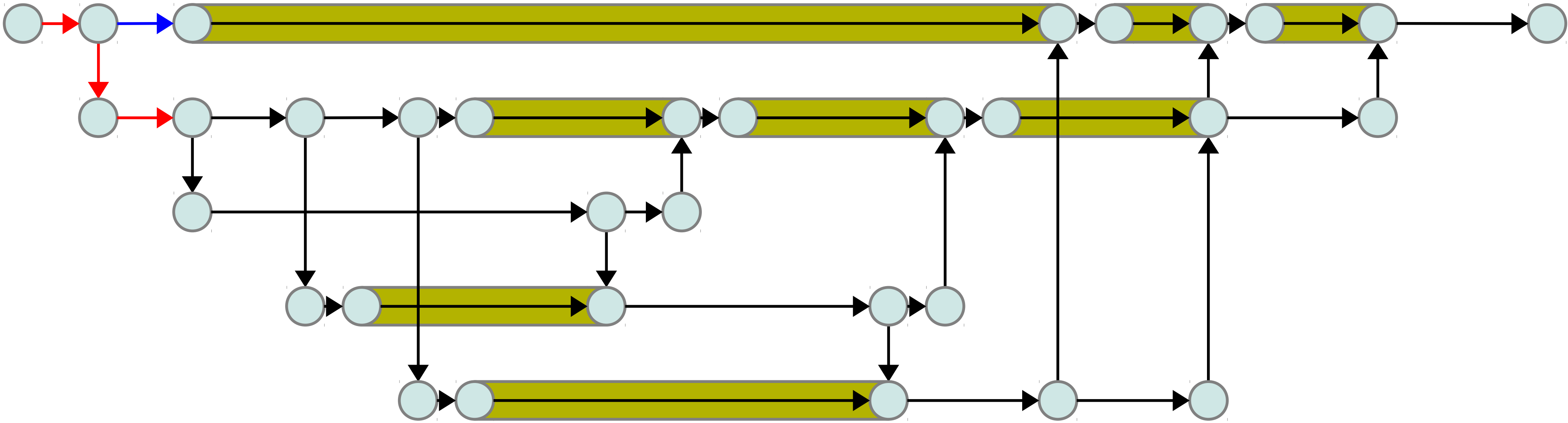


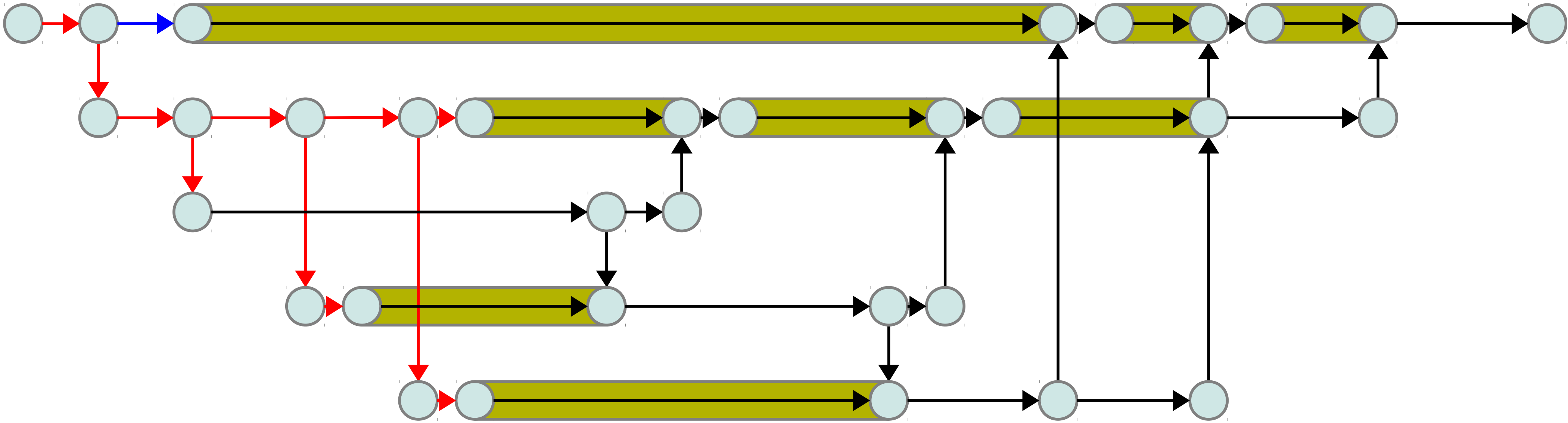


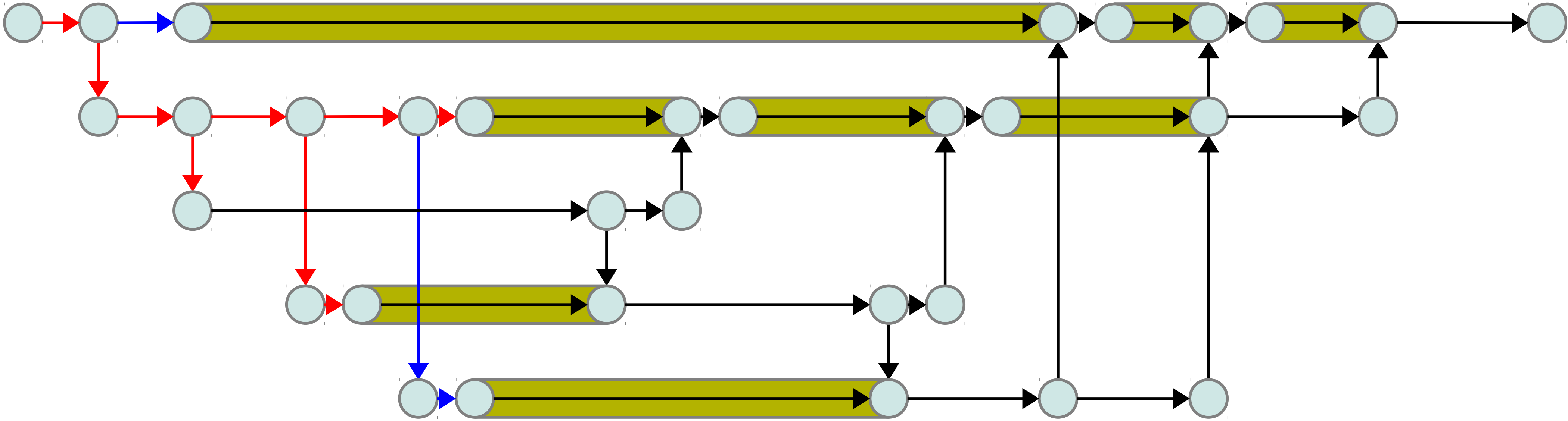


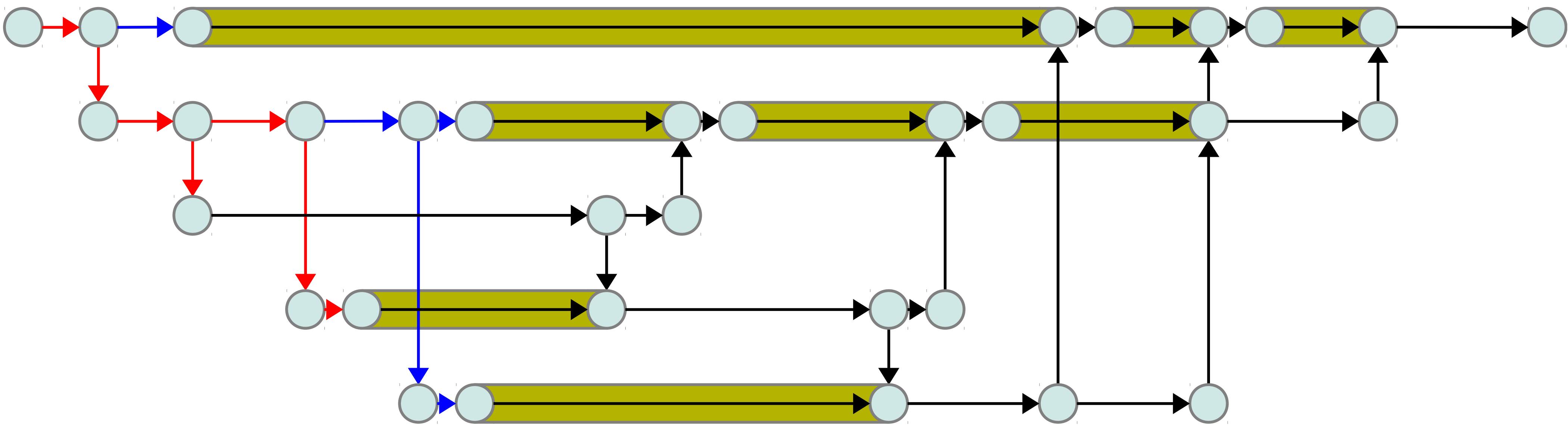


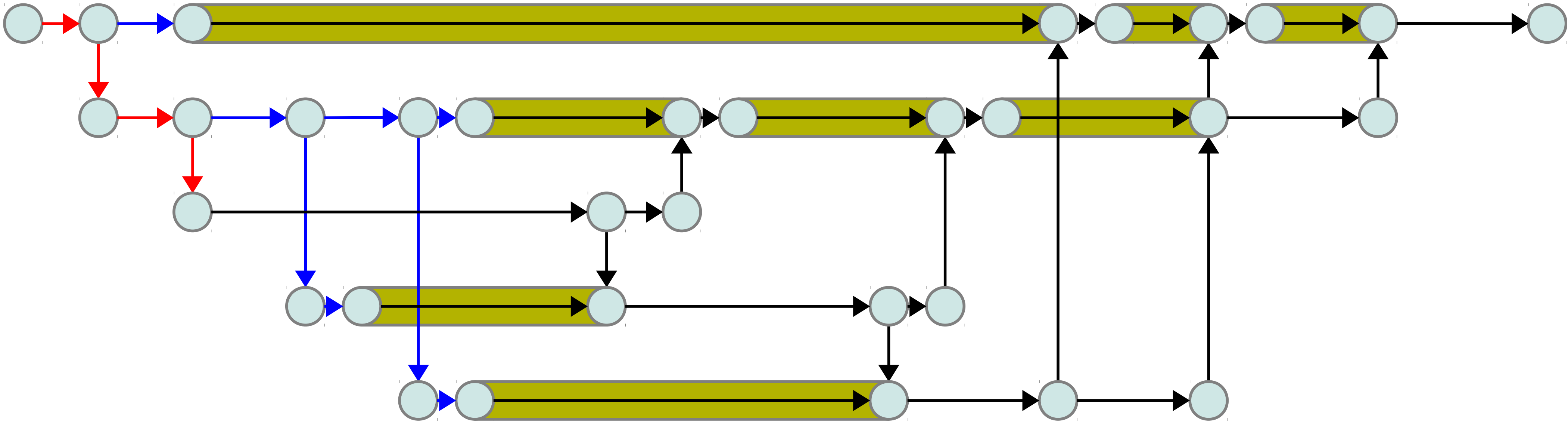


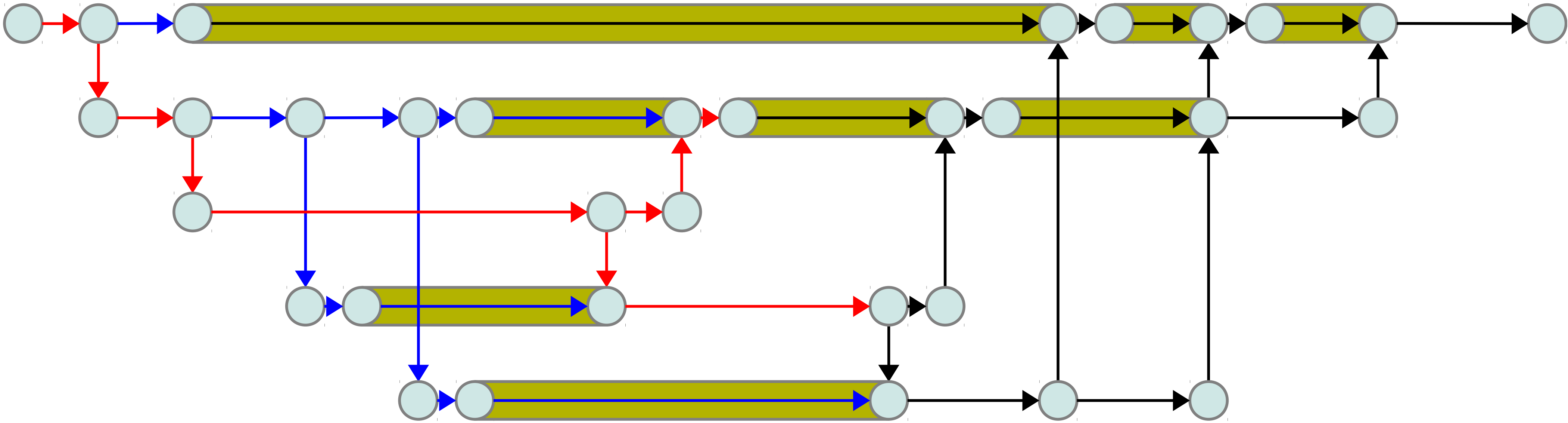


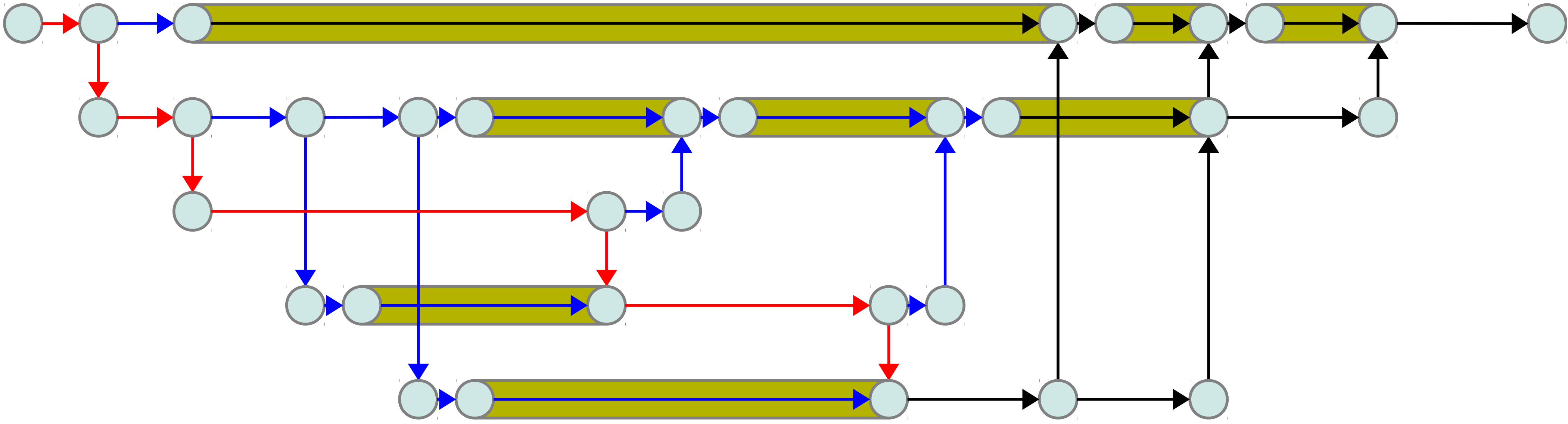


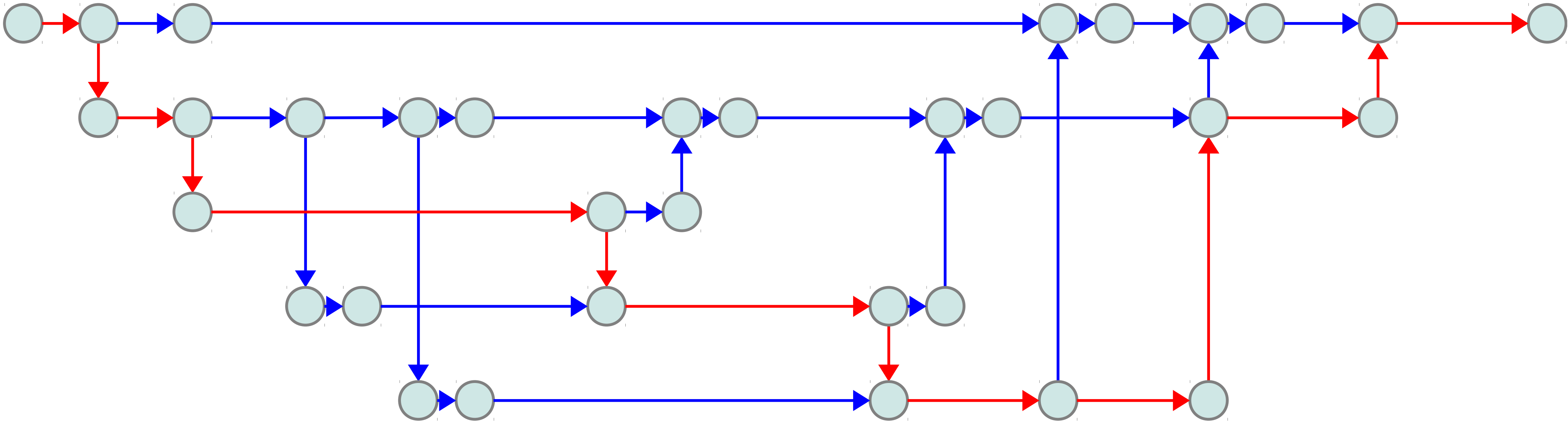


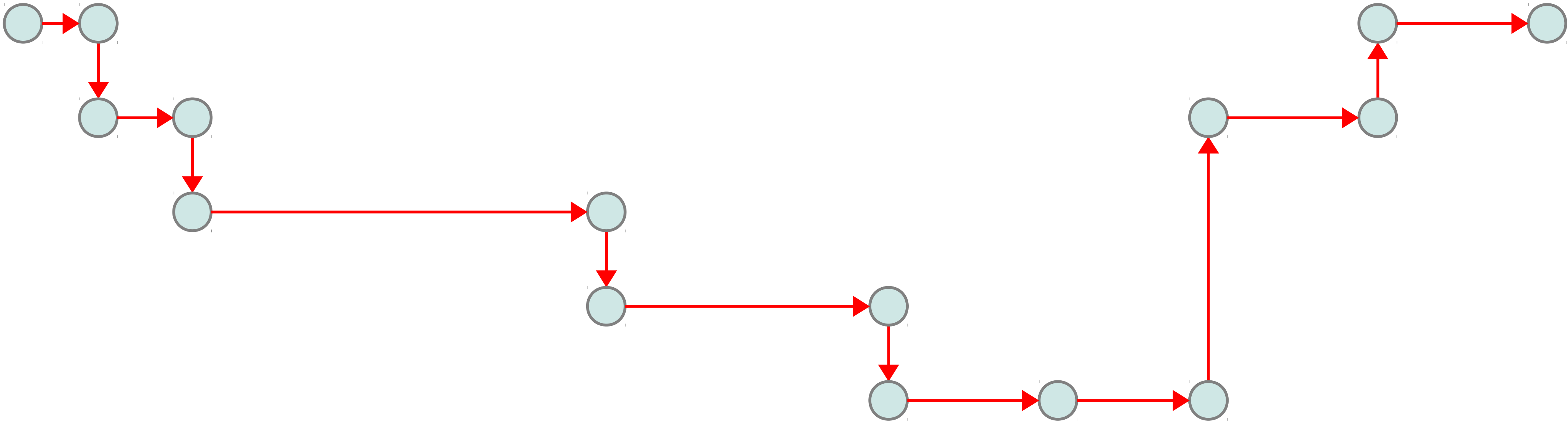


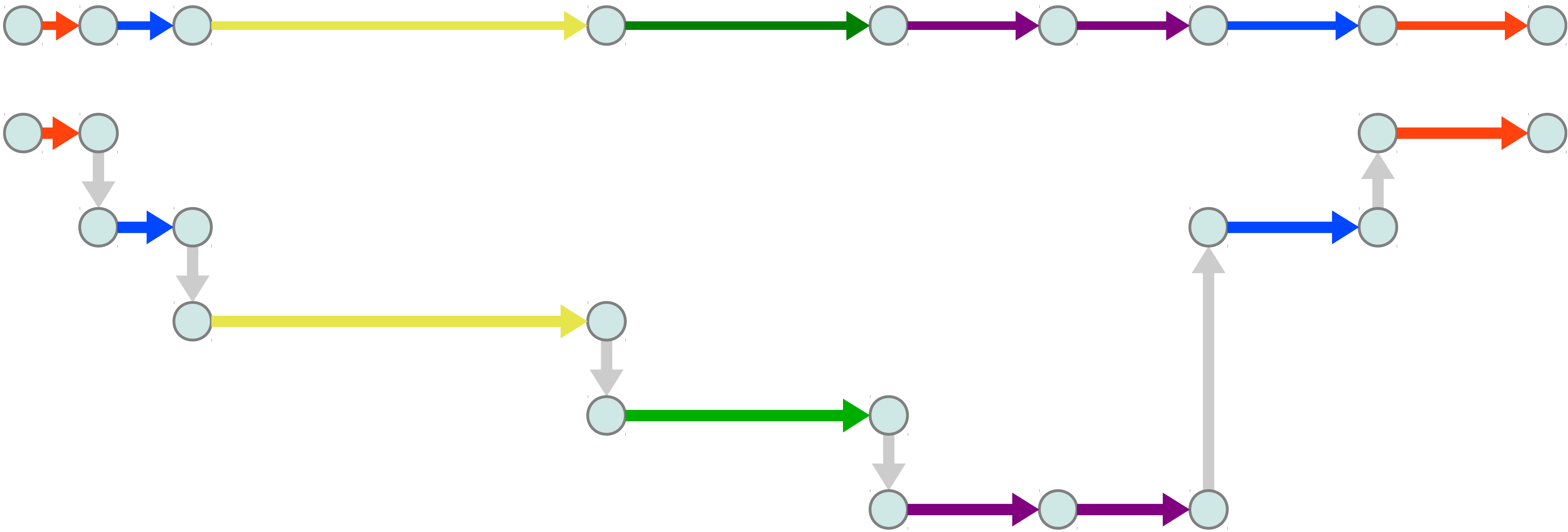




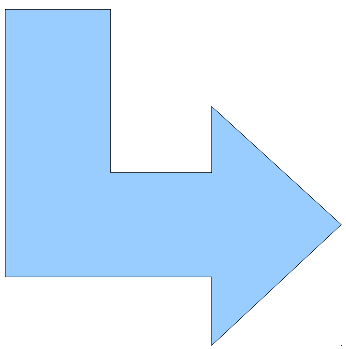








Final result



Incl.	Self	Called	~~Function~~ PID
100.00	0.01	(0)	1281 /bin/bash
52.30	0.01	1	1282 /bin/bash
51.84	0.03	1	1283 /bin/ls
51.74	0.04	1	1284 /bin/tail
44.73	0.38	1	1285 /bin/grep

Analysis of distributed processes

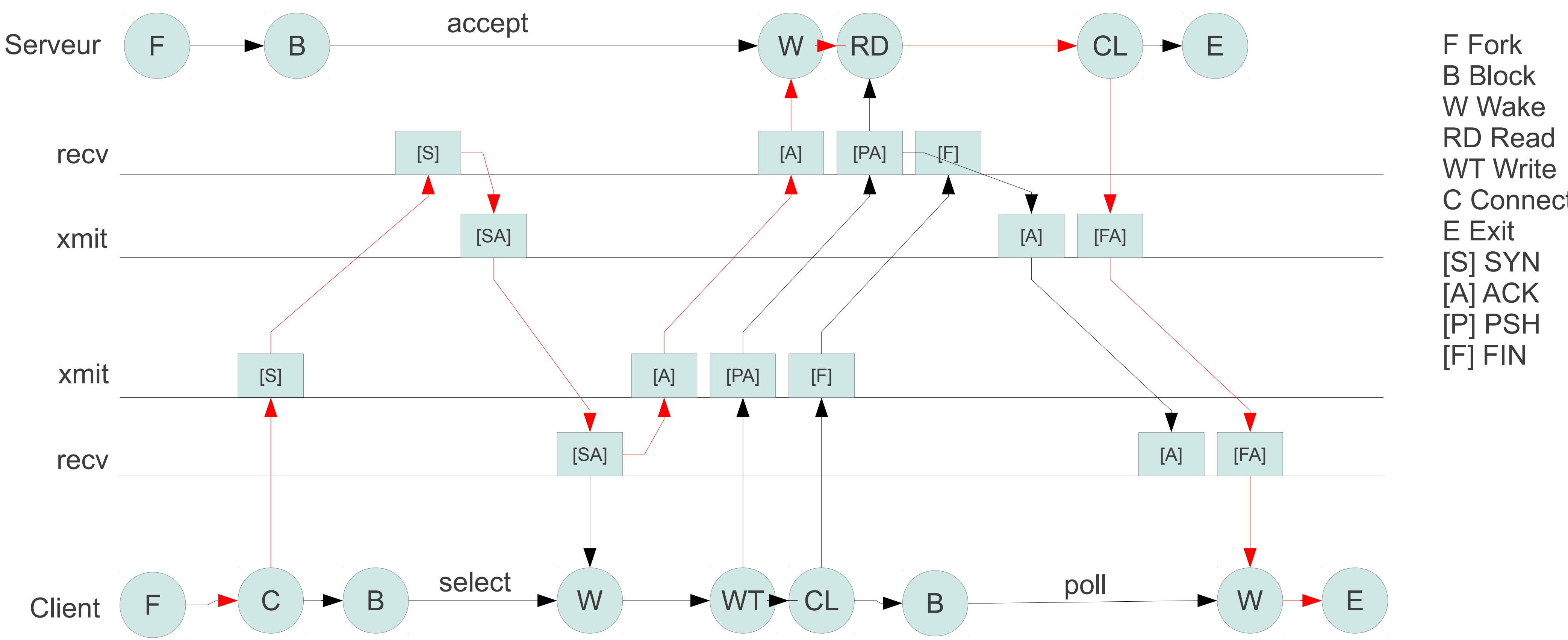
- Indirect wake-up from softirq
- Requires additional instrumentation
  - inet_sock_create
  - inet_sock_clone
  - inet_sock_delete
  - inet_sock_local_in
  - inet_sock_local_out

Overrides inet familly and uses kprobe hook

Netfilter hook

Critical path involving network I/O

echo “foo” | netcat under Traffic Shaper (slow-motion)



Future work

- Finish prototype implementation
- Visualization and summary of critical path
- Validation with real workload
- Multi-host analysis

Conclusion

- Research addresses real challenge
- Proposed approach is original
- Preliminary analysis shows the feasibility

Thanks to Professor Michel Dagenais and our partners.

CAE
The National Defense of Canada
Ericsson
Opal-RT
Révolution Linux

References available into the research proposal document.

Software:

http://secretaire.dorsal.polymtl.ca/~fgiraldeau/workload-kit/

http://secretaire.dorsal.polymtl.ca/~fgiraldeau/traceset/

https://github.com/giraldeau