System-level critical path analysis

Progress Report Meeting
May 2nd 2013

Francis Giraldeau
francis.giraldeau@polymtl.ca

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

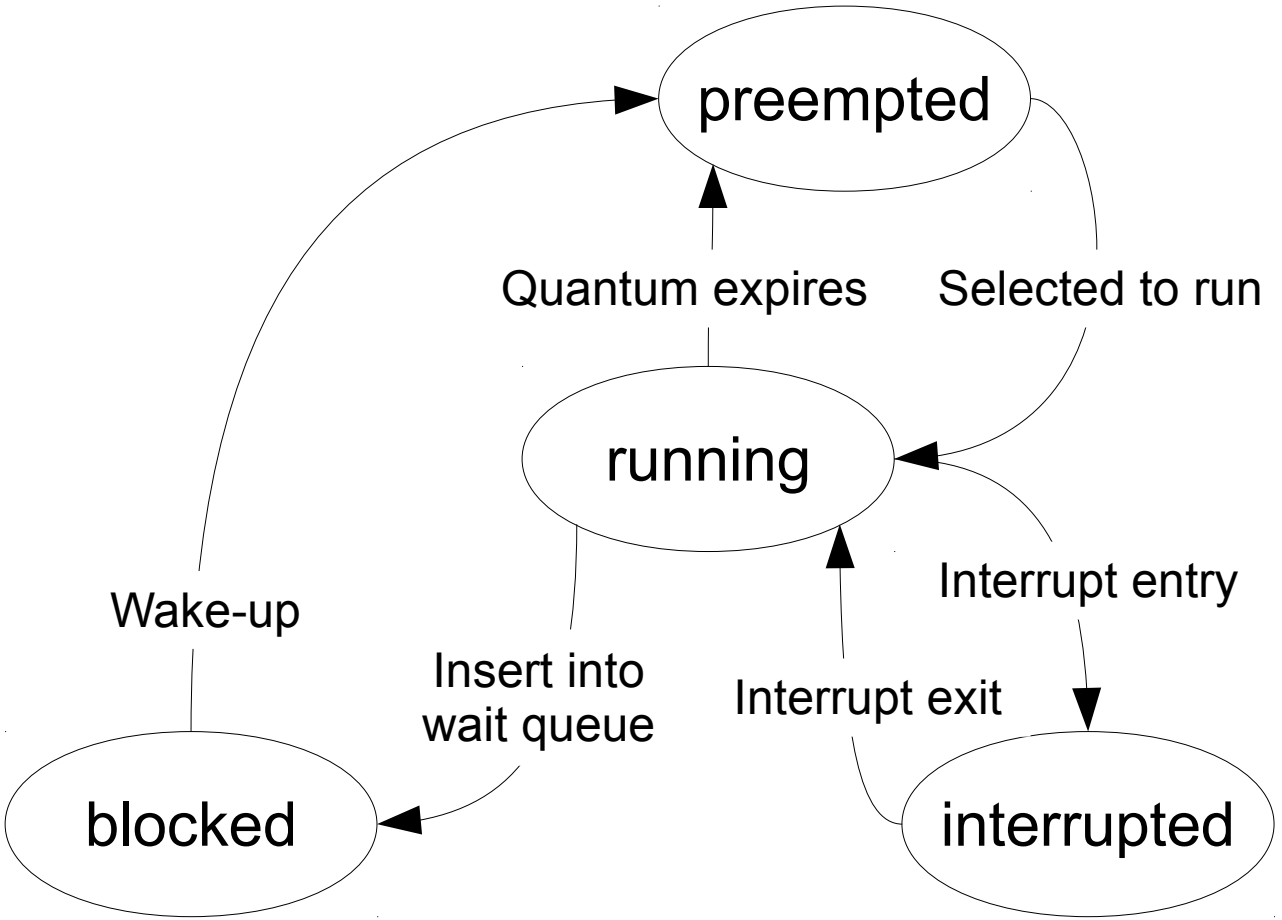
Plan

- Research objectives
- Build execution graph without system call
- Critical Path
  - Forward algorithm
  - Unbounded backward algorithm
  - Bounded backward algorithm
- Demo
- Future work

General objective

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

Task states

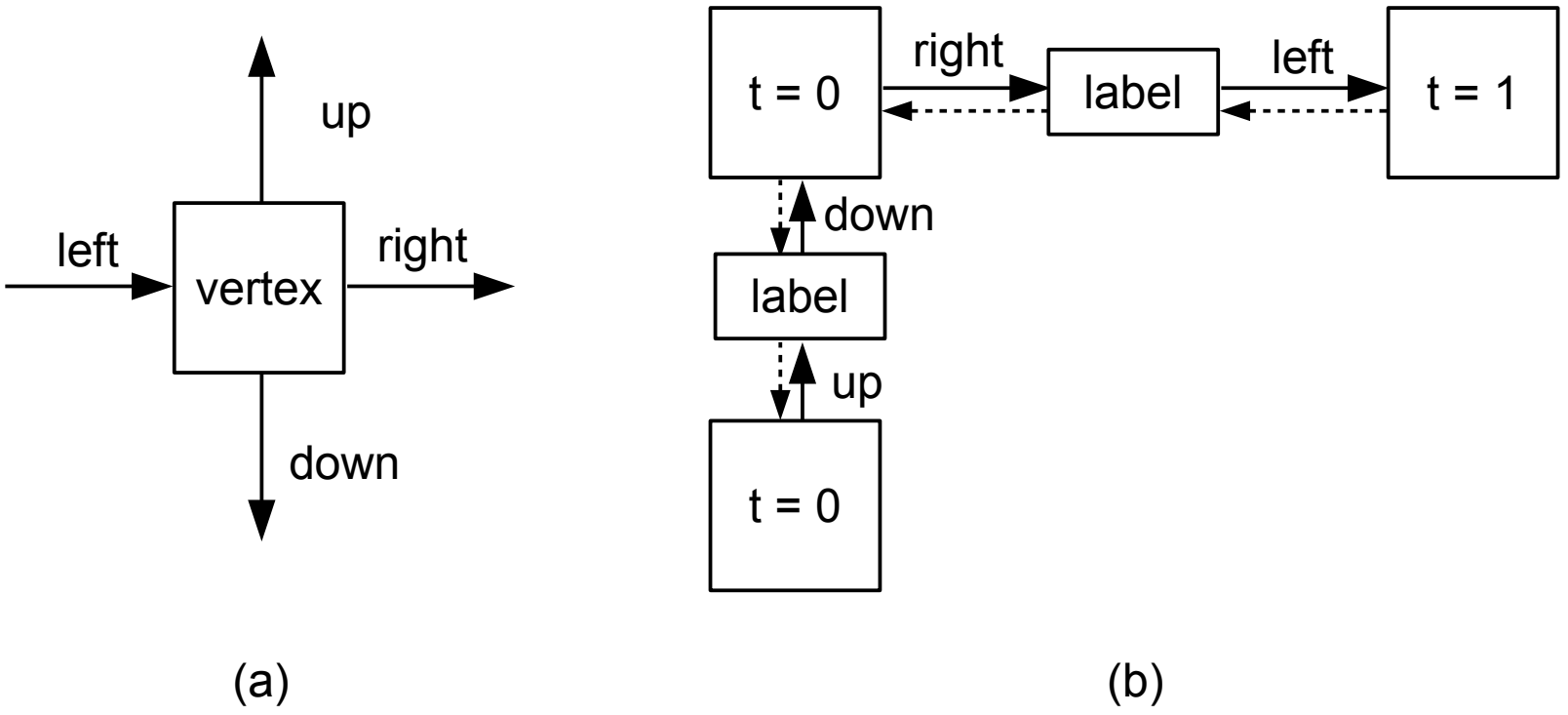

preempted
Quantum expires
Selected to run
running
Wake-up
Insert into
wait queue
Interrupt exit
Interrupt entry
blocked
interrupted

Required events

- sched_schedule
- sched_wakeup
- sched_wakeup_new
- irq_handler_entry/exit
- hrtimer_expire_entry/exit
- softirq_entry/exit

System calls not required!

Graph data structure



(a) (b)

Graph data structure (cont.)

```
public class Node {

    public static final int UP = 0;
    public static final int DOWN = 1;
    public static final int RIGHT = 2;
    public static final int LEFT = 3;

    public Link[] links;
    private long ts;

}
```

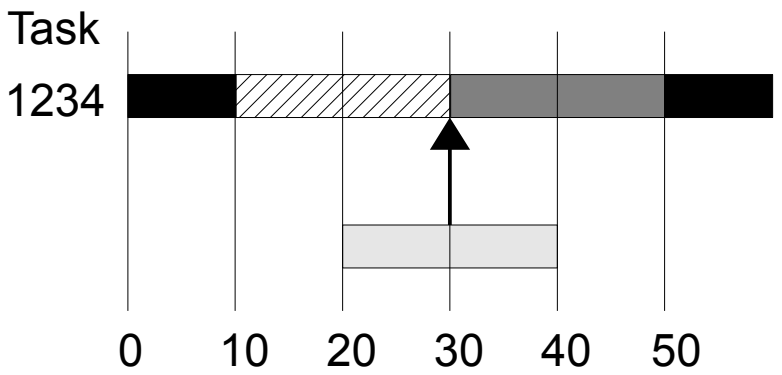
```
public class Link {

    public LinkType type;
    public Node from;
    public Node to;

}
```

```
public enum LinkType {

    EPS,
    UNKNOWN,
    DEFAULT,
    RUNNING,
    BLOCKED,
    INTERRUPTED,
    PREEMPTED,
    TIMER,
    NETWORK,
    USER_INPUT,
    BLOCK_DEVICE

}
```

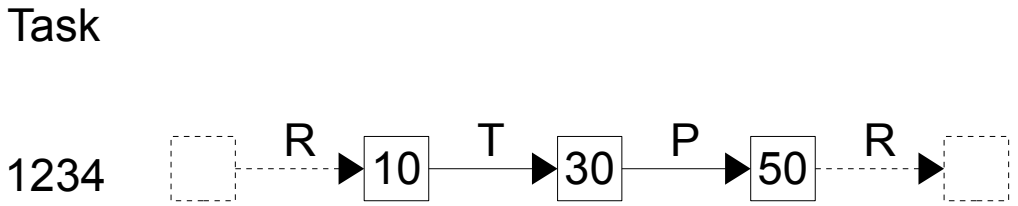
Execution graph construction (1)

(a)

Time	Event	CPU	Fields
10	sched_switch	0	{ prev_tid = 1234, prev_state = 1 }
20	hrtimer_expire_entry	1	{ }
30	sched_wakeup	1	{ tid = 1234 }
40	hrtimer_expire_exit	1	{ }
50	sched_switch	0	{ next_tid = 1234 }



(b)

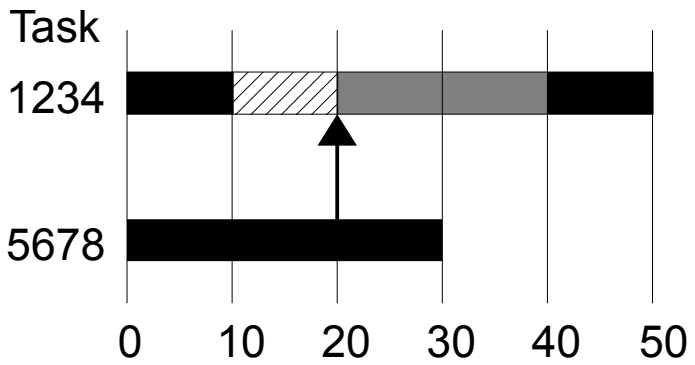


(c)

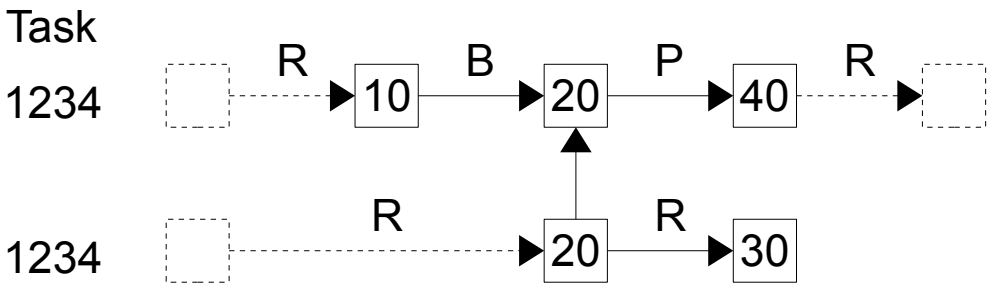
Execution graph construction (2)

(a)

Time	Event	CPU	Fields
10	sched_switch	0	{ prev_tid = 1234, prev_state = 1 }
20	sched_wakeup	1	{ tid = 1234 }
30	sched_switch	1	{ prev_tid = 5678 }
40	sched_switch	0	{ next_tid = 1234 }



(b)



(c)

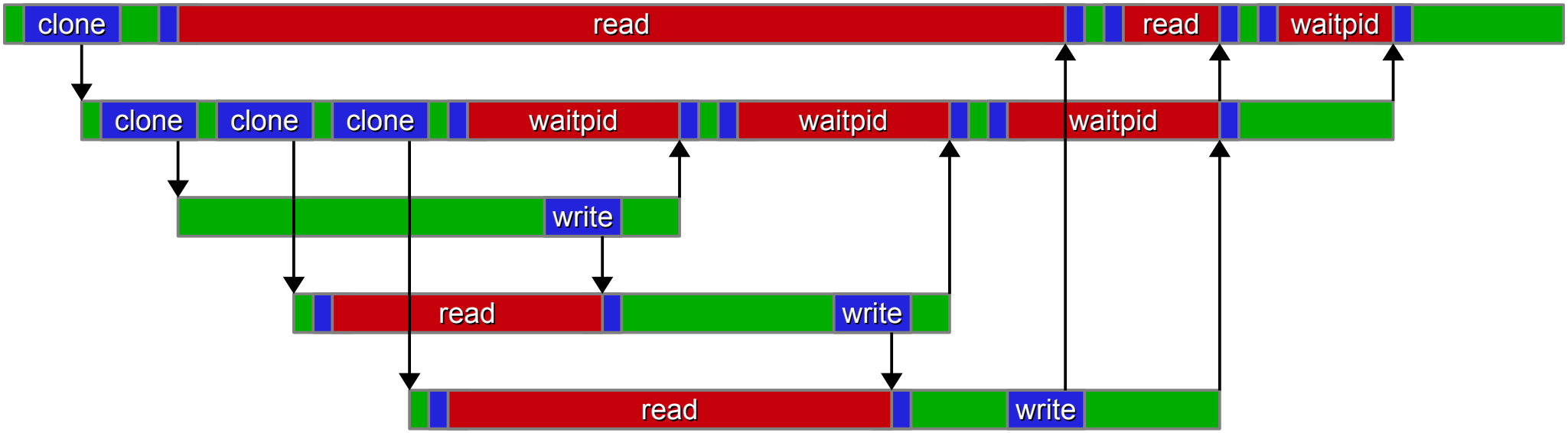
Critical path computation

- Forward speculative algorithm
- Forward unbounded backtracking
- Forward bounded backtracking

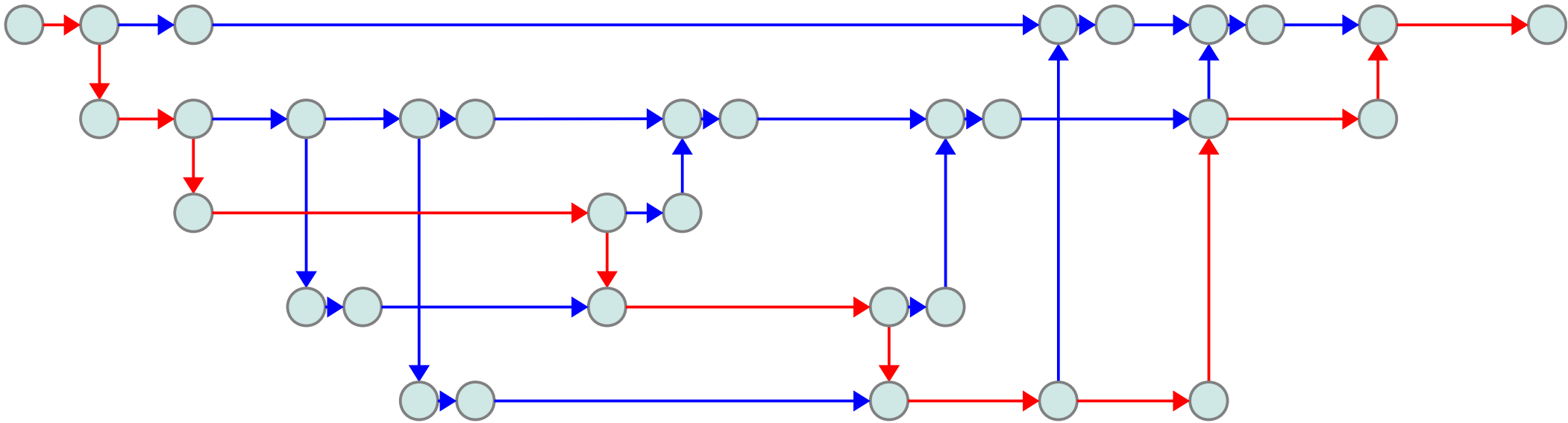
5 processes are involved:

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

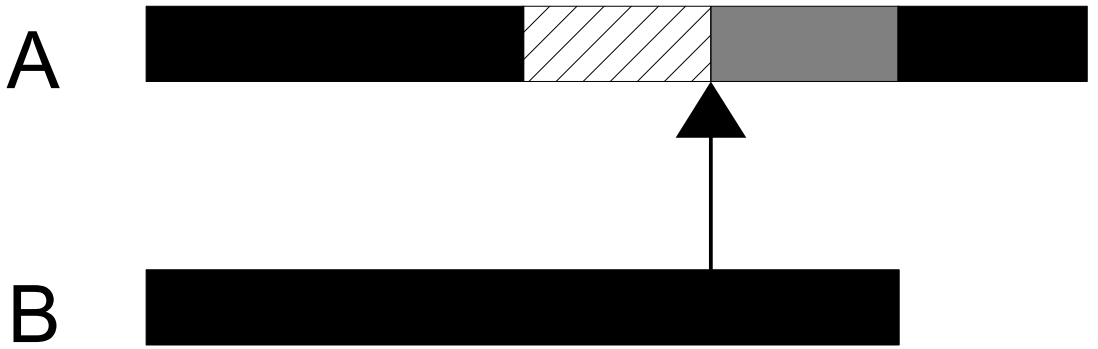
Forward speculative algorithm



Forward speculative algorithm

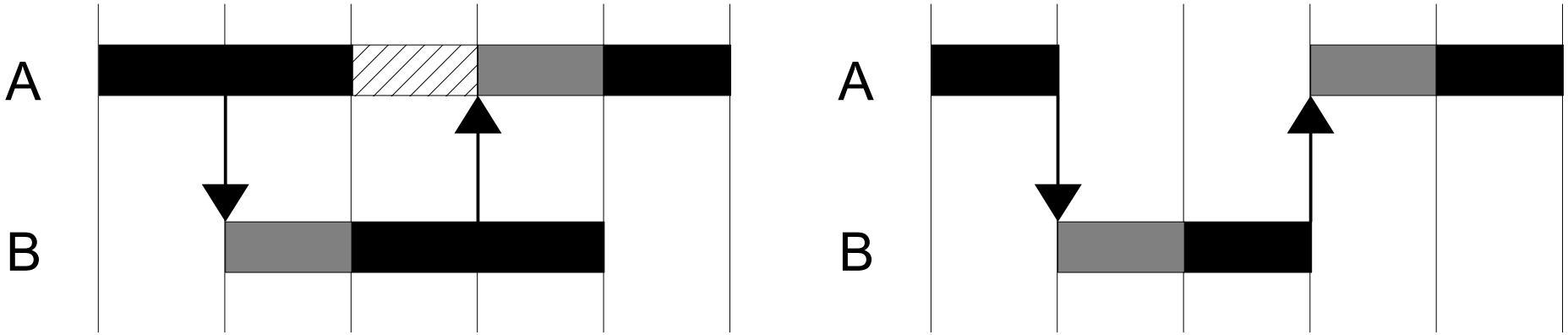


Problem with siblings

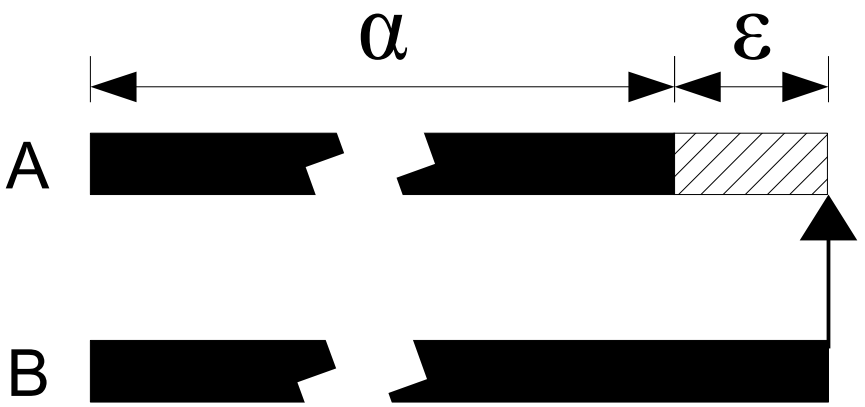


Empty critical path!

Unbounded algorithm



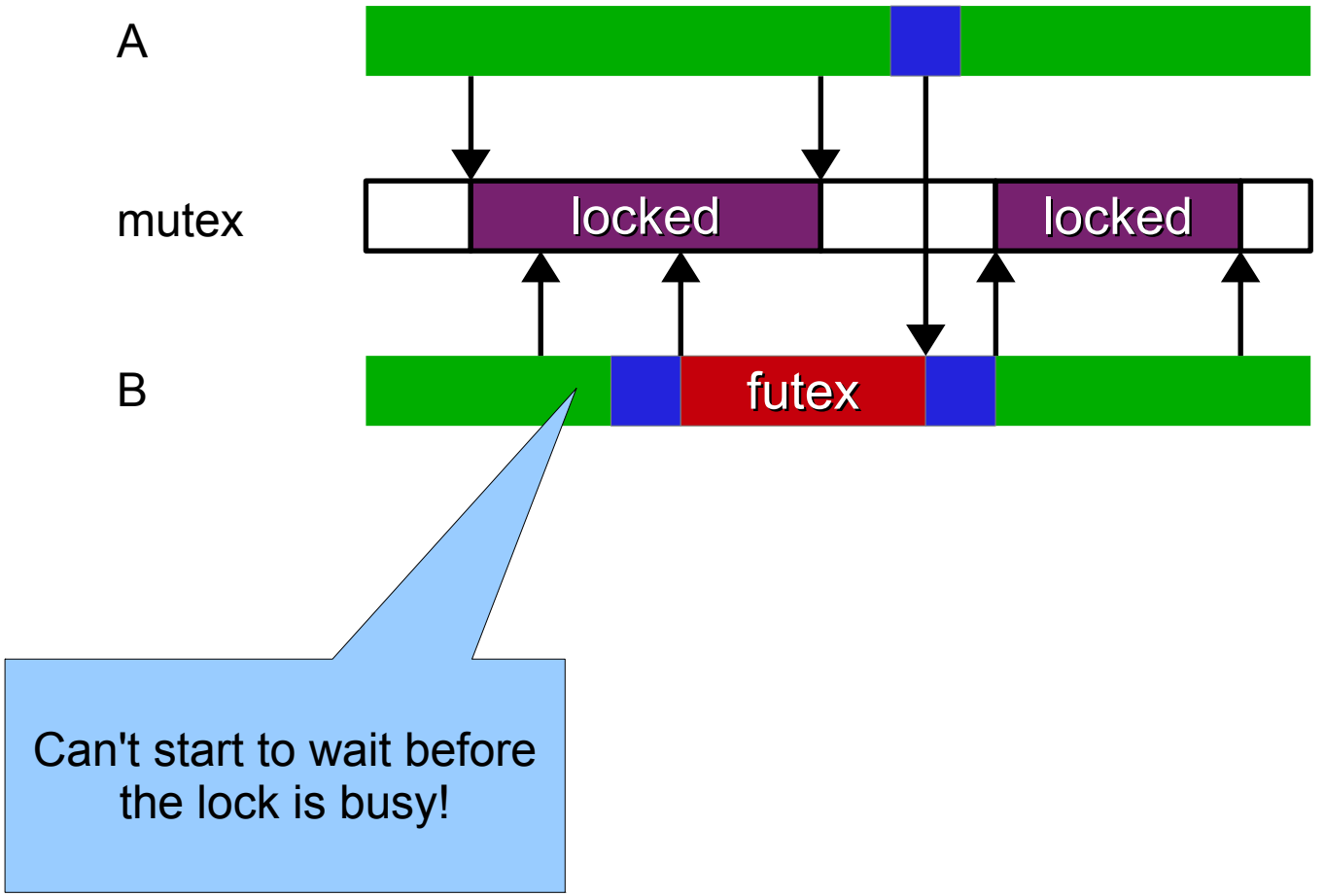
Problem of unbounded backtrack



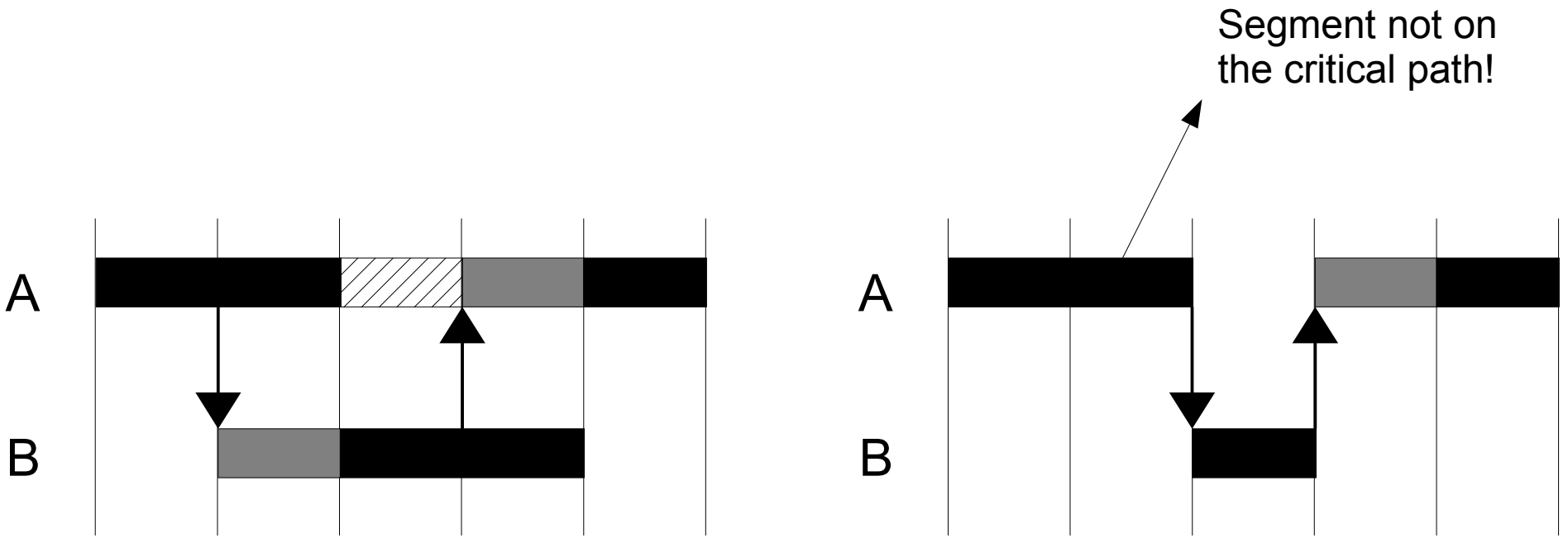
Let $\alpha >> \varepsilon$ and $P = 0.5$

The critical path WRT task A is either ~A or ~B.

Use case: futex

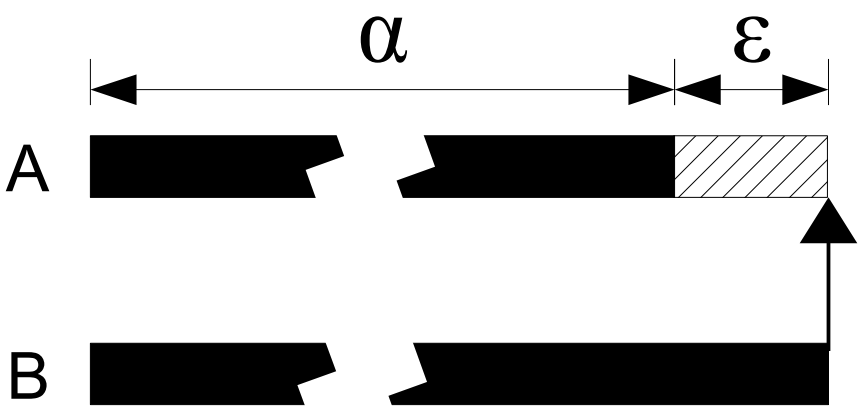


Bounded algorithm



Bounded algorithm is an approximation of the critical path

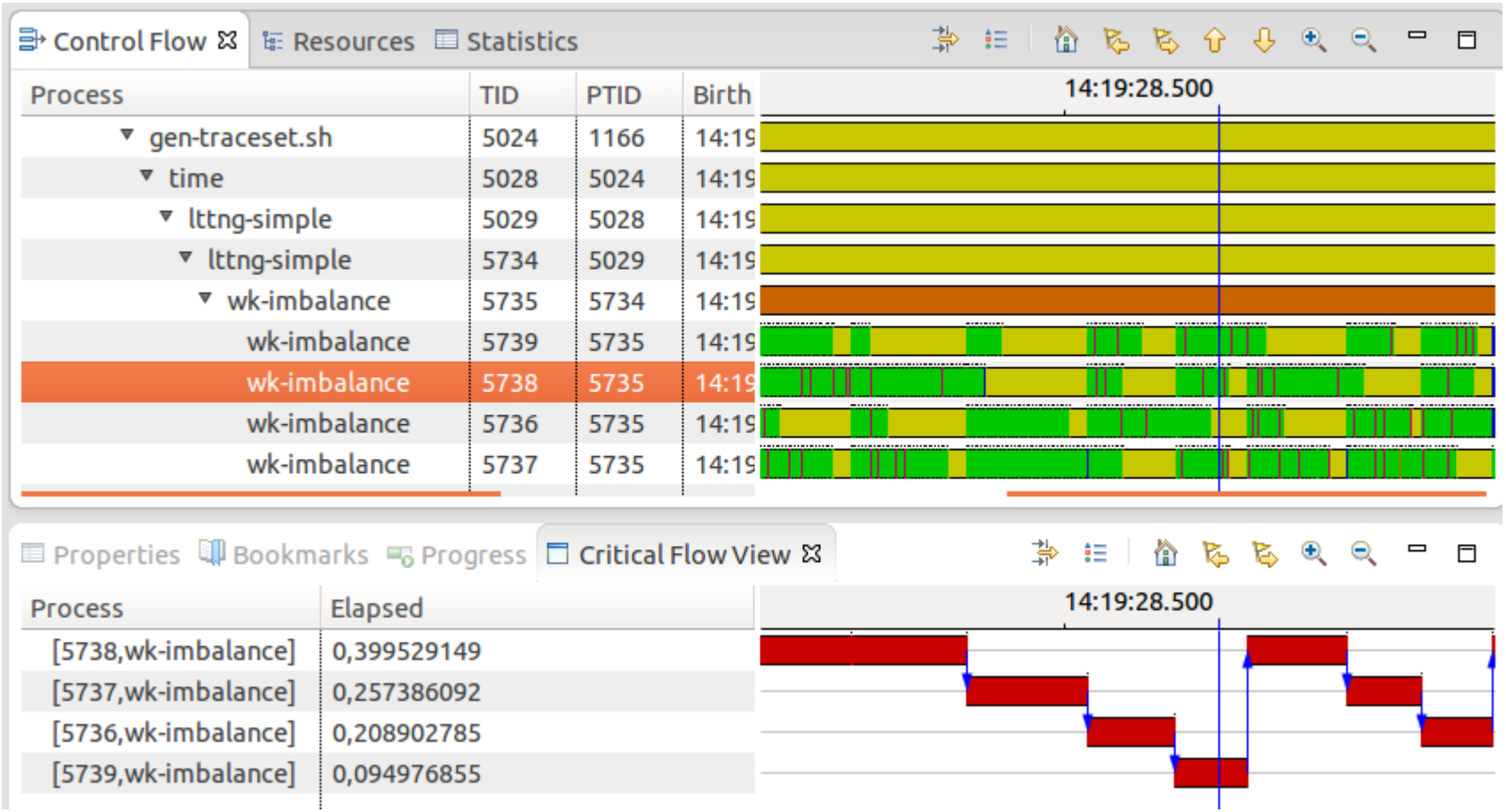
Result of bounded backtrack

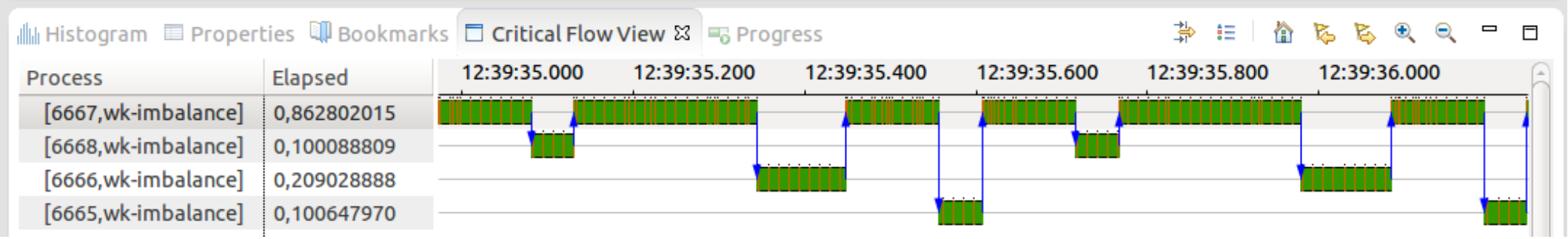


Let $\alpha >> \varepsilon$ and $P = 0.5$

The critical path WRT task A is always ~ A.

Example with wk-imbalance



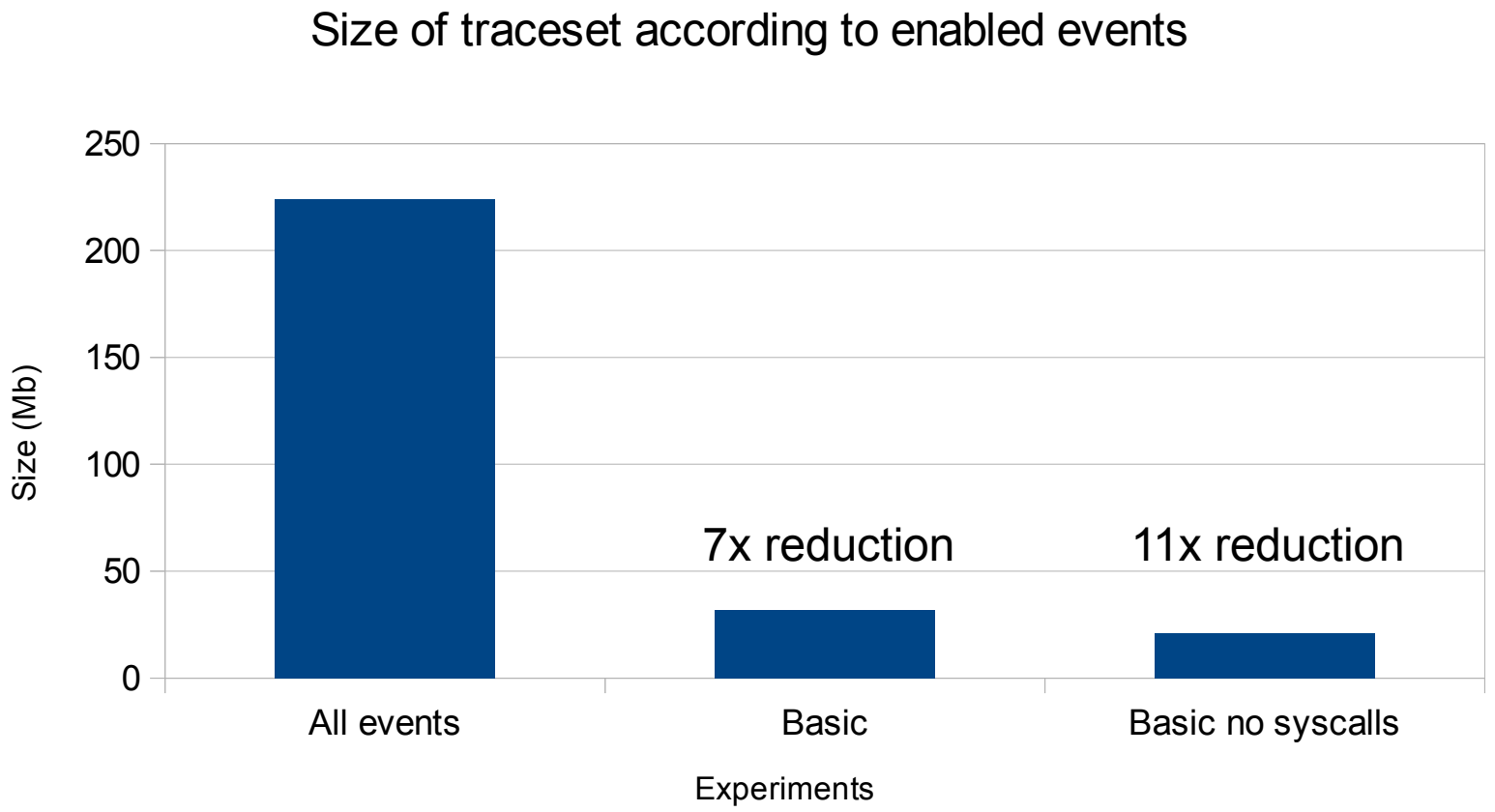


Demo

Future work

- Quantify the error of bounded algorithm
- Select bounded or unbounded algorithm
- On-line computation
- Multi-host analysis
- SHT for memory scalability
- Bridge the gap between kernel trace and userspace code
- Reduce overhead with specialized tracepoints

Reducing overhead



Conclusion

- Proposed method allows to observe user and kernel tasks
- Unbounded algorithm has limitations
- Approximation of bounded algorithm produces stable results, but introduces some error
- Works on real world examples

Thanks to Professor Michel Dagenais and our partners EfficiOS and Ericsson.

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