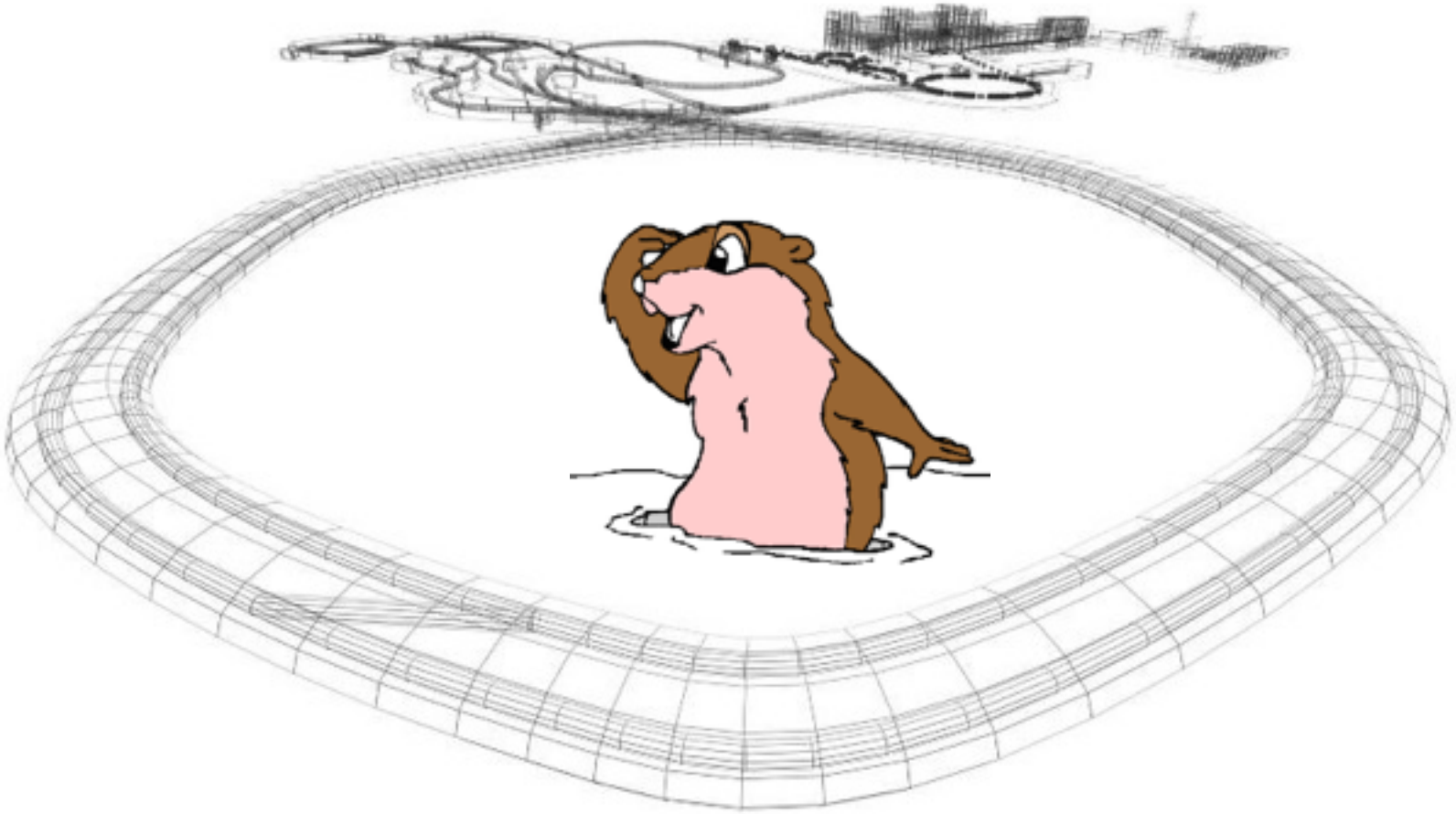
Inexpensive Scheduling in FPGAs

W. Terpstra, M. Kreider, D. Beck



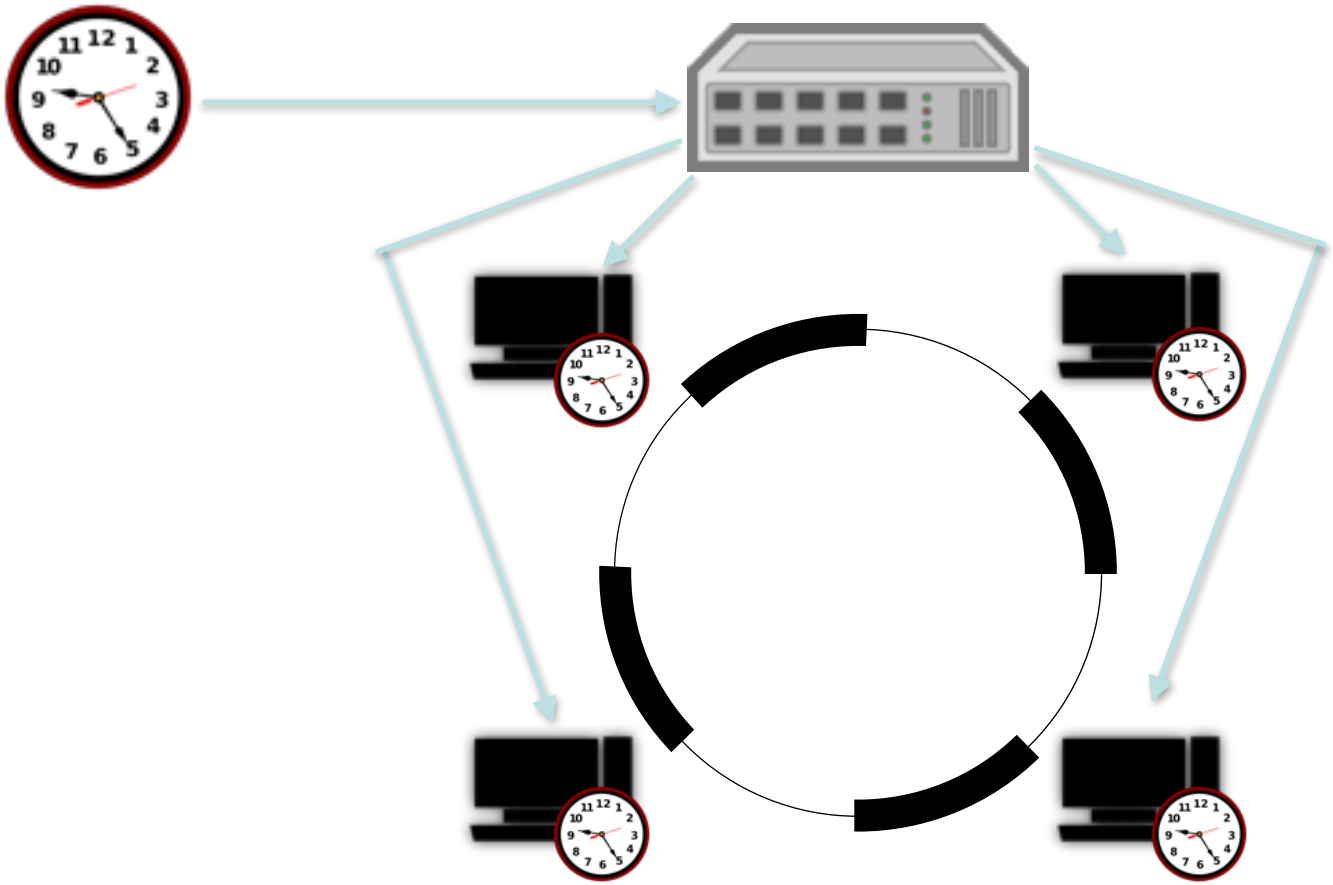
What to expect

- Scheduling: executing planned events on time

- This Talk: a trick to schedule **very** cheaply in hardware
  … by exploiting the real-time requirement

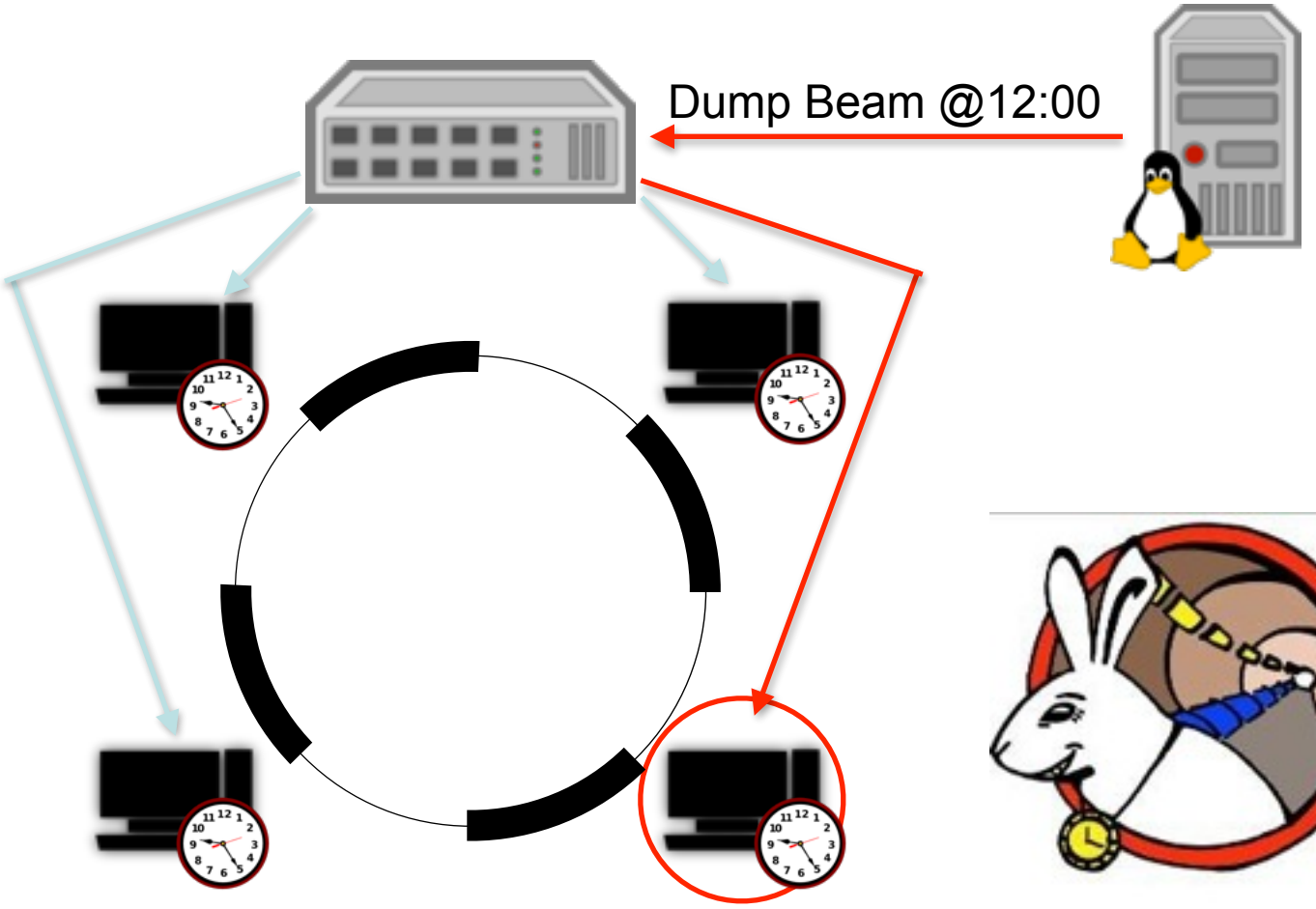
Context: Globally known time

- White Rabbit delivers accurate time (<1ns) to all controllers



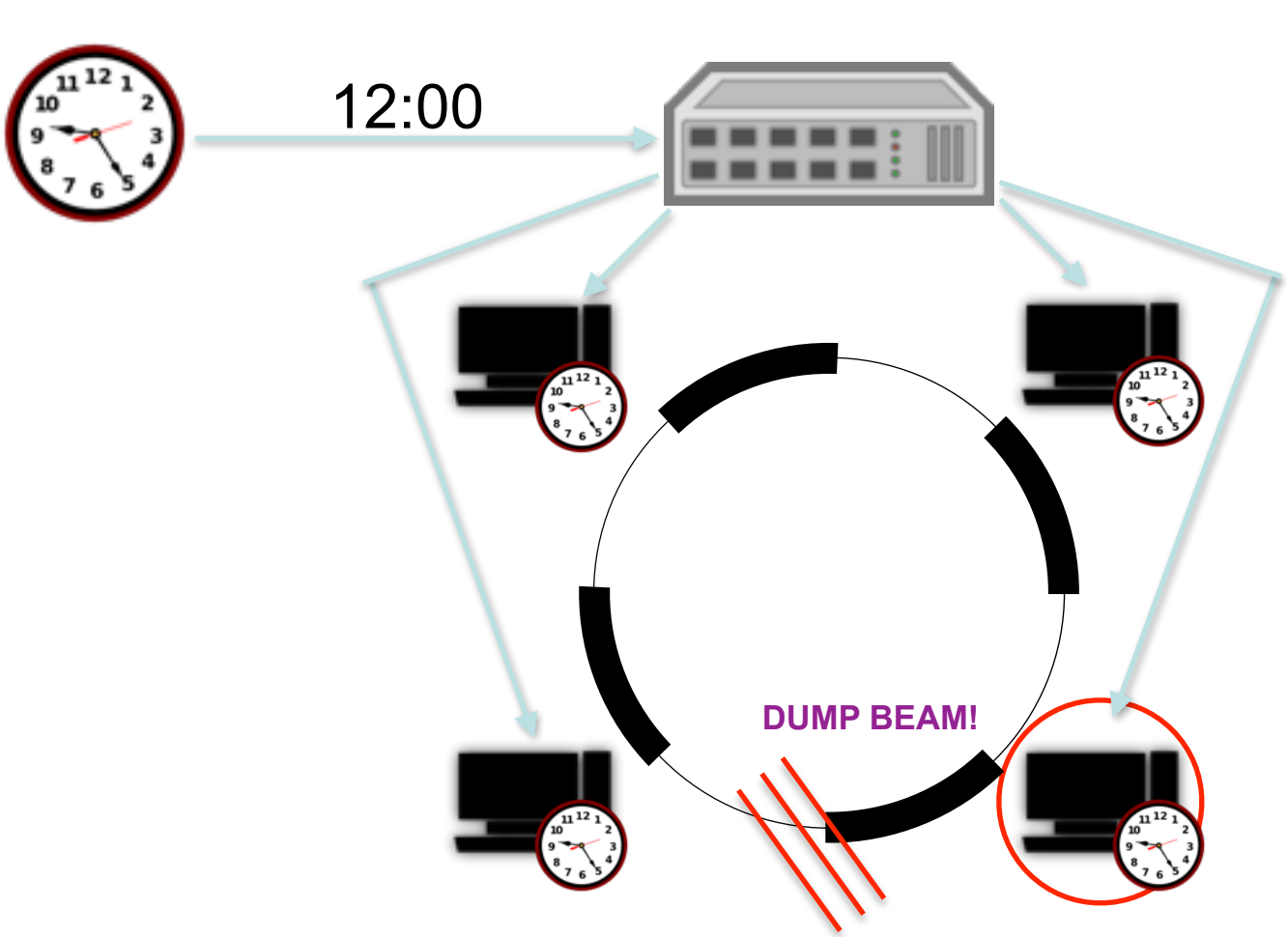
Context: DM says what to do

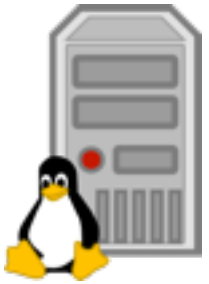
- Data-Master says what to do and when to do it



Context: Execute actions on time

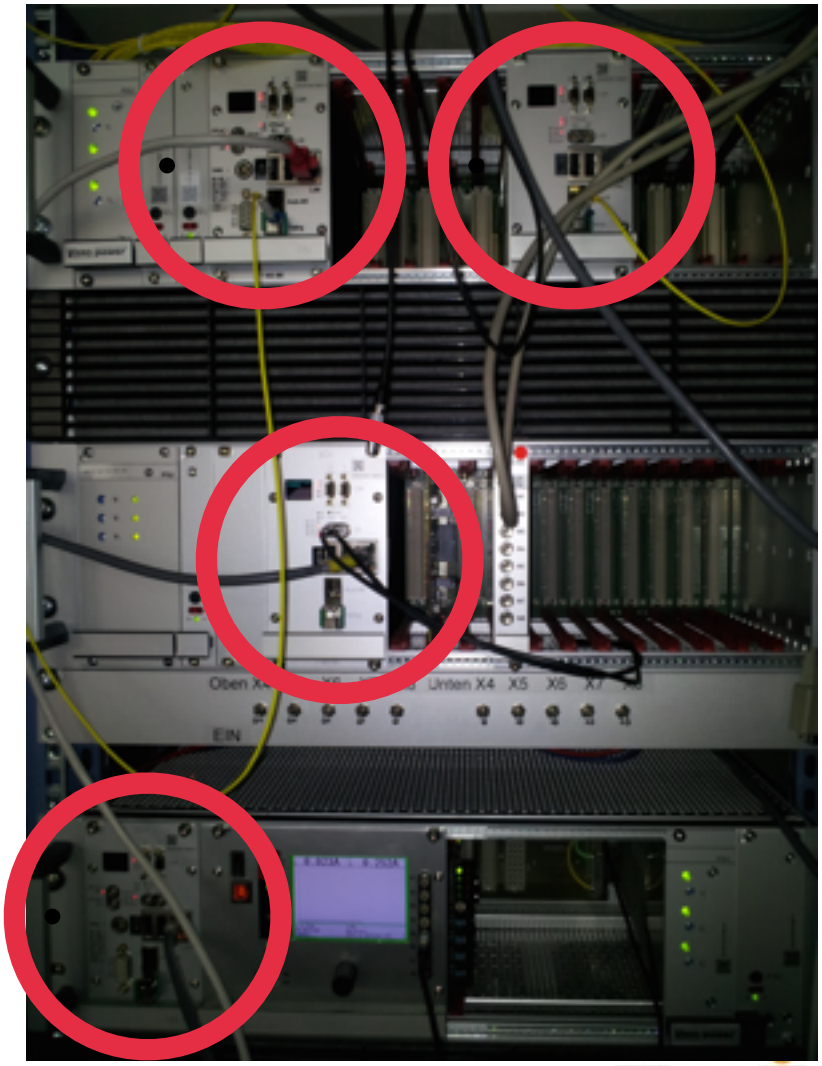
- When the scheduled time is reached, action occurs





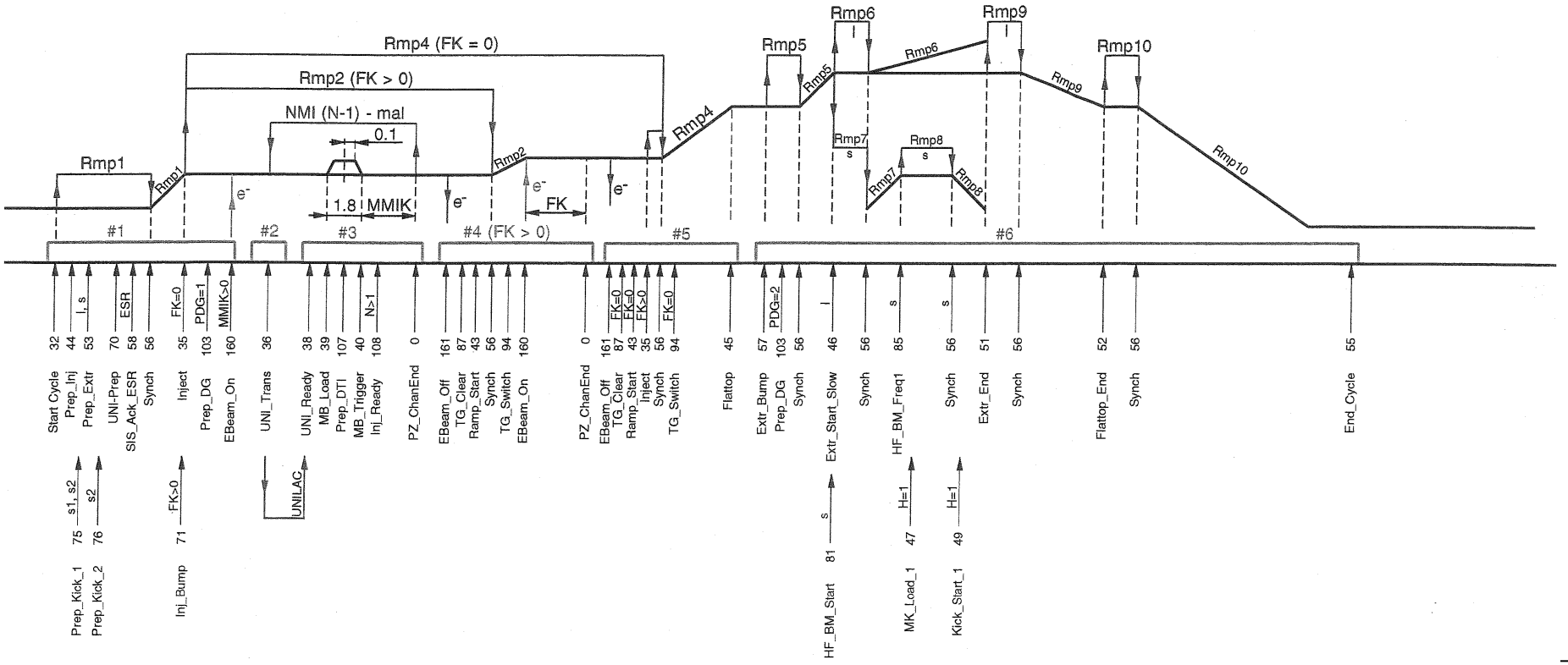
Reality: Front-end Controllers

- Front-end Controllers (FECs) actually look something like:
- and there are thousands…
- and they include FPGAs



Reality: Ramping a Magnet

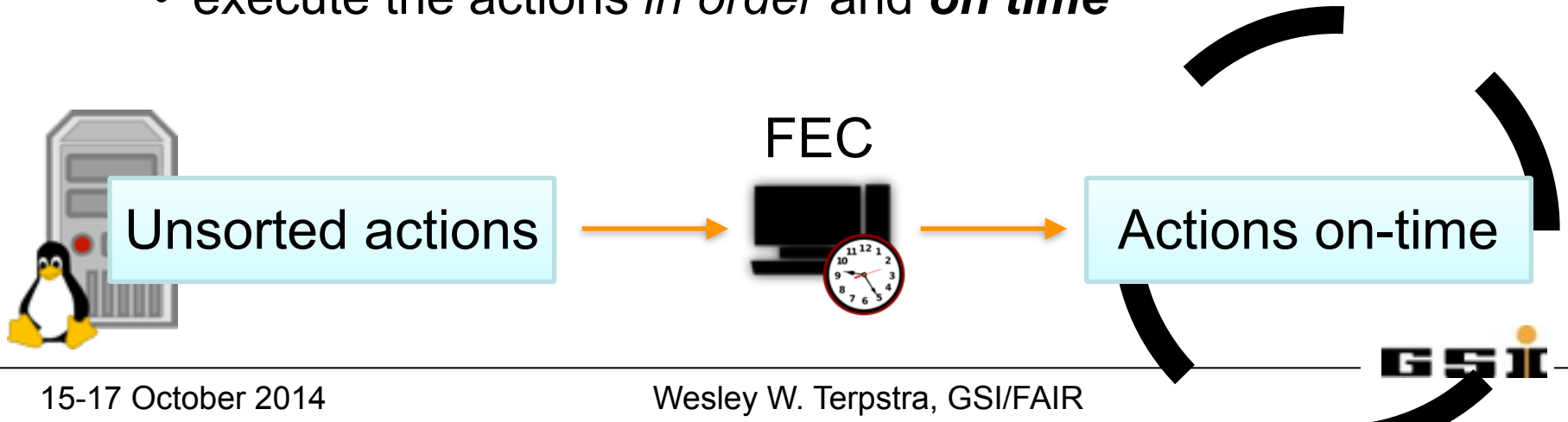
- Real action execution looks something like



The Problem: Scheduling Events

- Main point: DM sends actions to take before FECs take them
  … slight problem: the actions do not arrive in order

- Problem to solve:
  - receive actions at FECs *out-of-order*
  - execute the actions *in order* and ***on time***



But: Sorting?!

- Scheduling is at least as hard as sorting
  - Proof: schedule input #s as events and pop them in-order
- Sorting requires log(n) comparisons per element
- Can solve directly
  - Heaps: priority queues / heap-sort
  - Implemented in VHDL by M. Kreider (see his poster)
- But! There is a loop-hole in the sorting complexity proof:
  - log(n) “comparisons” can be: read the bits of the timestamp
  - This talk => fitting an elephant through that loop-hole

Calendars: Mankind's O(1) scheduling

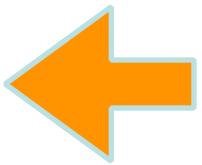
- DM tells you to do something on March 12th?
  - write that into your calendar
- Every day when you wake up
  - check the calendar
  - do whatever due that day
- Avoids log(n) cost?

  Bits of timestamp =

  index into calendar

Calendars: Snake Oil?

- **Seems too easy! You cheated.**
- Well, there are two caveats:
  - Scheduling usually solves a harder problem
    - Find the ***next*** task vs. find **today**'s task



Key

  - The calendar is very big

    2^64 entries for FAIR

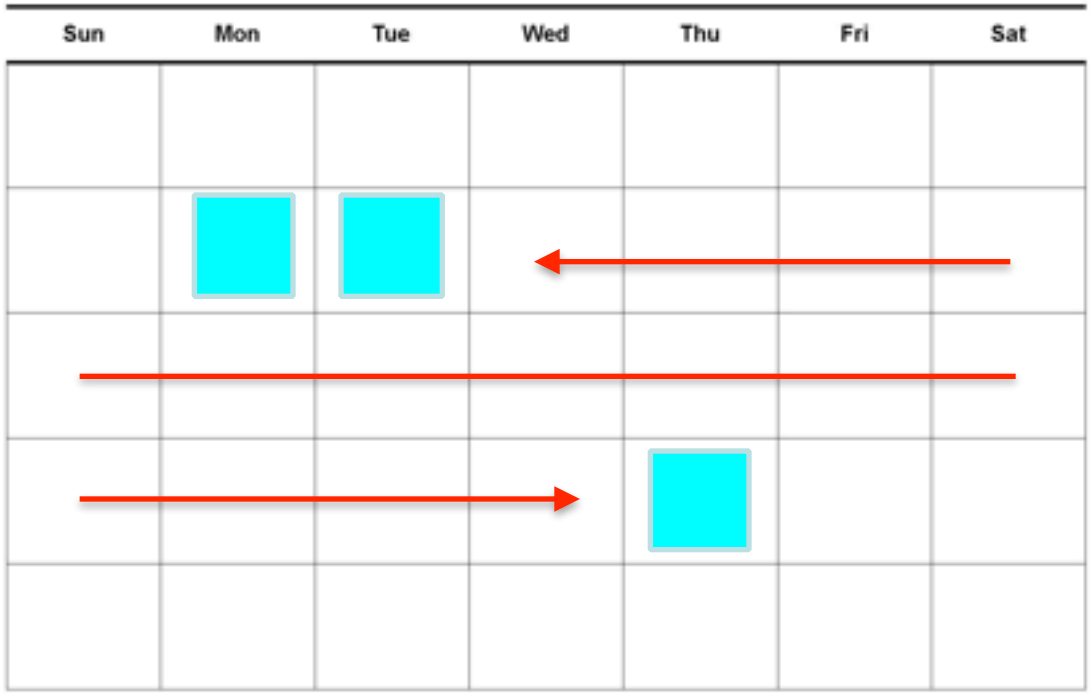
- To schedule a real-time accelerator we only need **today**'s task

Calendars: Empty holes

- Calendar-based scheduling means skipping over holes
  - time spent inspecting empty days is wasted time

- Bucket-sort
- Radix-sort
- SW calendar queues
  all perform poorly for non-uniform schedule density

- Real-time / FPGA?
  no extra cost



Why checking today is enough

- Hardware is composed of busy wait loops
  - ➡ must make a decision on every rising clock edge

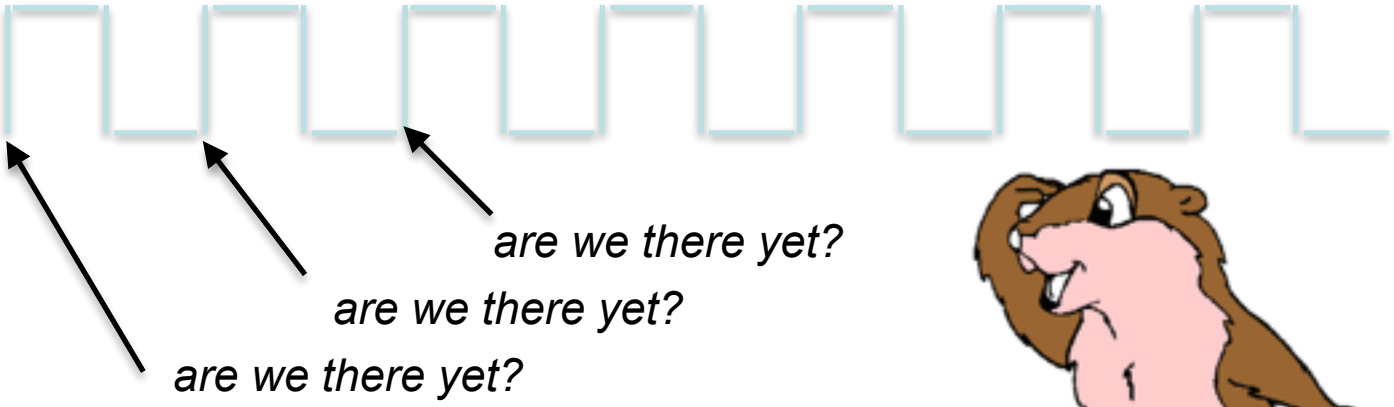
are we there yet?

are we there yet?

are we there yet?

Why checking today is enough

- Hardware is composed of busy wait loops
  - ➡ must make a decision on every rising clock edge

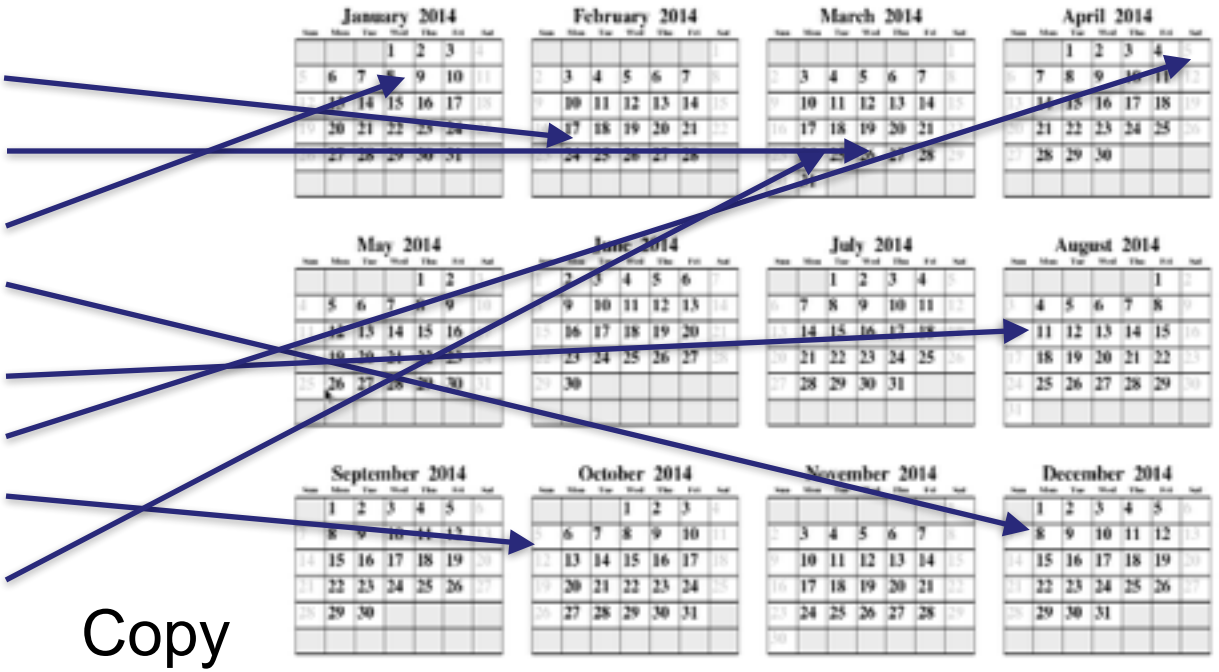


- If you know the ***next*** event is at time n
  - You could ask, “Is t=n yet?” on every rising edge
  - But, you might as well ask, “Is Calendar[t] set?”
  - ➡ Knowing n does not make the problem easier!

The remaining problem

- Still: Calendars are too big!
  - Small (<< 2^64 actions) original problem
  - Expanded into giant (= 2^64 entries) calendar

0	Feb 24	beep
1	Mar 19	blink
2	Jan 8	on
3	Dec 25	off
4	Aug 11	flash
5	Apr 5	echo
6	Oct 12	ping
7	Mar 24	pong

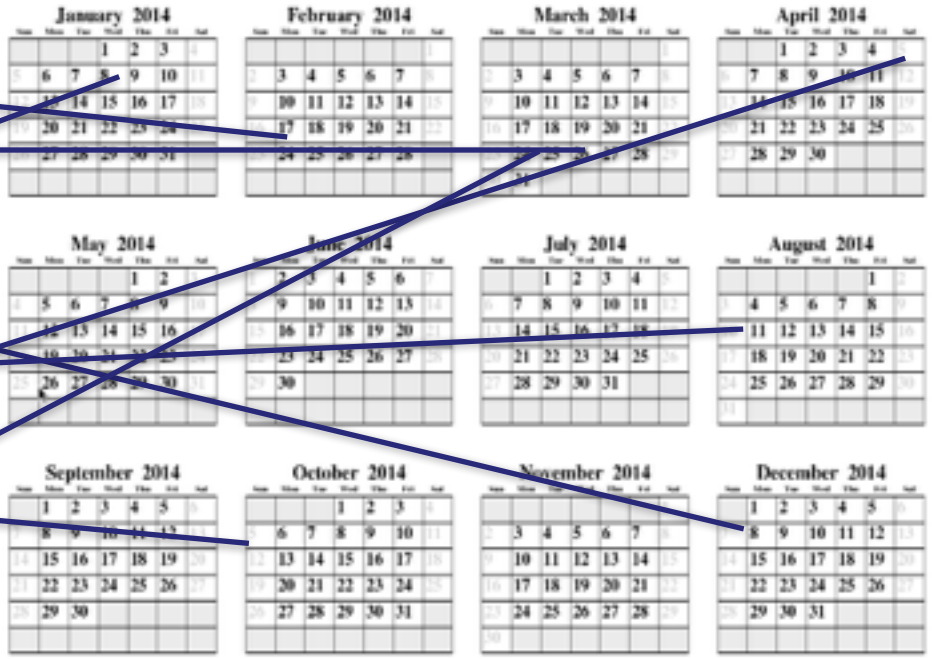


Copy

The remaining problem

- A small improvement
  - Don’t copy action (big) into calendar entries
  - Instead: just store a reference (small) to unsorted table

0	Feb 24	beep
1	Mar 19	blink
2	Jan 8	on
3	Dec 25	off
4	Aug 11	flash
5	Apr 5	echo
6	Oct 12	ping
7	Mar 24	pong



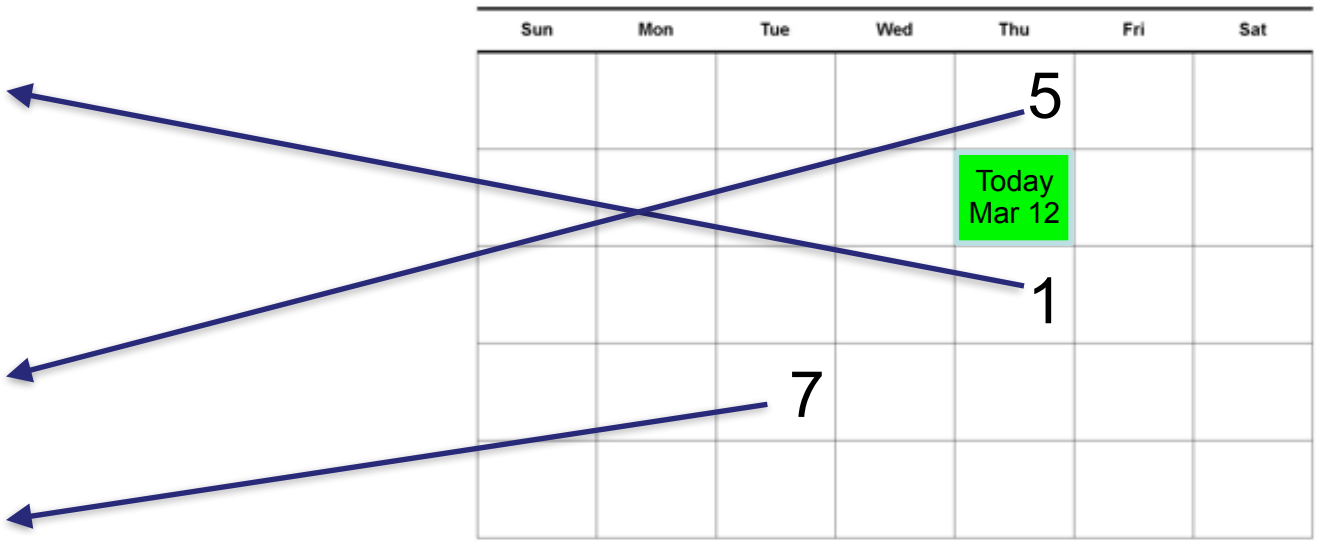
Pointer

The remaining problem

- The trick: just use a small calendar!
  - only keep track of a few days after today

- Just one small problem: not all actions are listed in calendar!

0	Feb 24	beep
1	Mar 19	blink
2	Jan 8	on
3	Dec 25	off
4	Aug 11	flash
5	Apr 5	echo
6	Oct 12	ping
7	Mar 24	pong



Pointer

Democracy

- Politicians: only interested in problems < 4 years away
- Public: cares about all problems
  - ➡ Regularly reminds politicians about unresolved problems
- Eventually every problem is < 4 years away
  - ➡ Eventually the public reminds a politician currently in office
  - ➡ Eventually a politician takes action on every problem

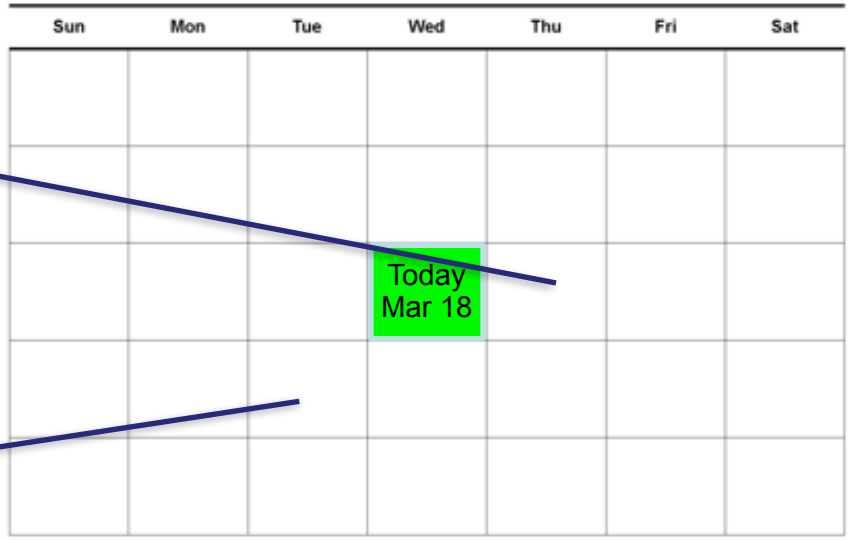
➡ Democracy works! (for all problems solvable in < 4 years)

Democratic Calendar Queues

- Two processes:
  - Check today's actions in calendar
  - Check if next unsorted action is < 35 days away

0	Feb 24	beep
1	Mar 19	blink
2	Jan 8	on
3	Dec 25	off
4	Aug 11	flash
5	Apr 5	echo
6	Oct 12	ping
7	Mar 24	pong

> today+35: no-op



Democratic Calendar Queues

- Two processes:
  - Check today's actions in calendar
  - Check if next unsorted action is < 35 days away

0	Feb 24	beep
1	Mar 19	blink
2	Jan 8	on
3	Dec 25	off
4	Aug 11	flash
5	Apr 5	echo
6	Oct 12	ping
7	Mar 24	pong

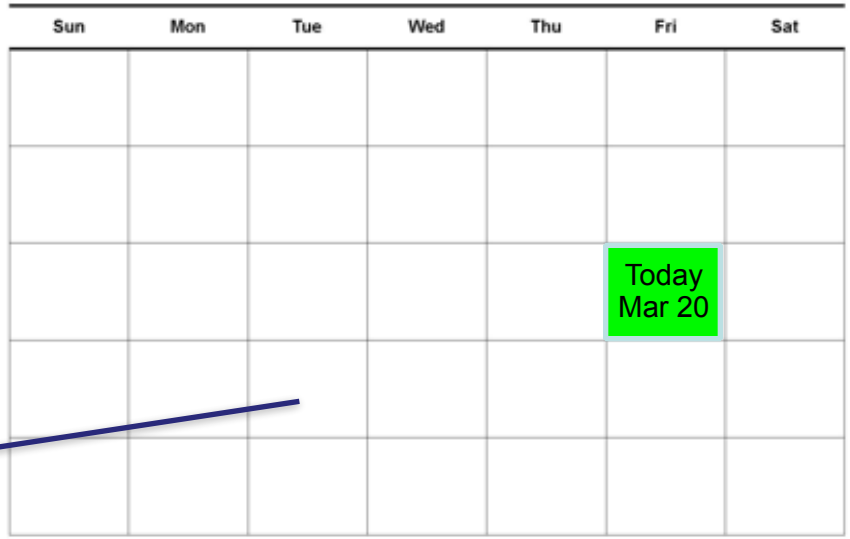
> today+35: no-op

Democratic Calendar Queues

- Two processes:
  - Check today's actions in calendar
  - Check if next unsorted action is < 35 days away

0	Feb 24	beep
1		
2	Jan 8	on
3	Dec 25	off
4	Aug 11	flash
5	Apr 5	echo
6	Oct 12	ping
7	Mar 24	pong

< today+35! update!

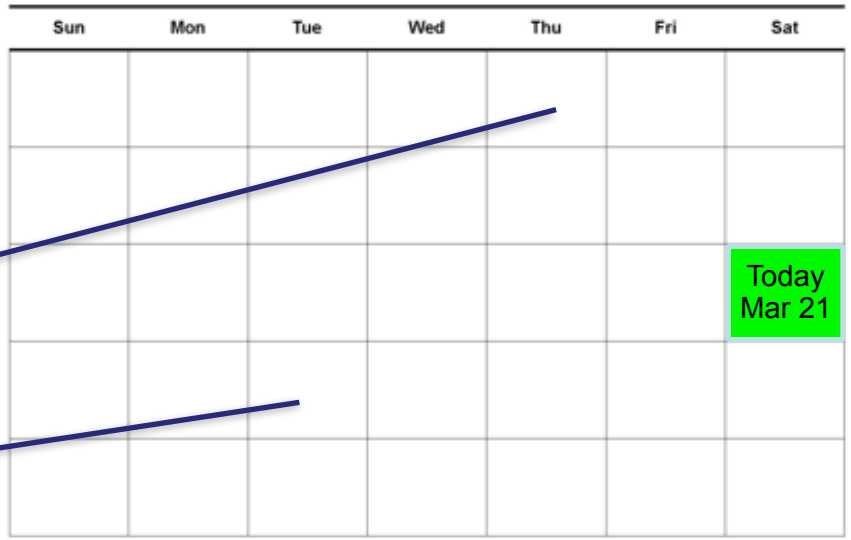


Democratic Calendar Queues

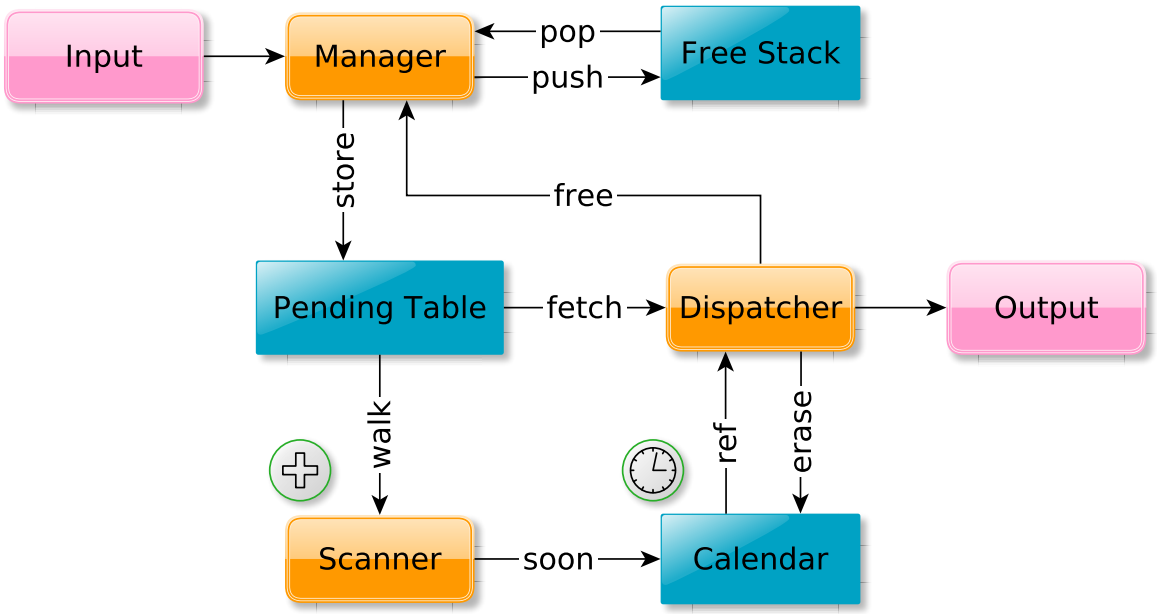
- Two processes:
  - Check today's actions in calendar
  - Check if next unsorted action is < 35 days away

0	Feb 24	beep
1		
2	Jan 8	on
3	Dec 25	off
4	Aug 11	flash
5	Apr 5	echo
6	Oct 12	ping
7	Mar 24	pong

> today+35: no-op



Block Diagram



- For a proof of when the democratic approach never fails
- For an explanation of how the components fit together

  … please read the full paper

Conclusion

- Democratic scheduling can be done very cheaply
  - O(1) area and time
  - As simple as it gets: 2 parallel memory accesses/cycle
  - Distribution of the actions in time is irrelevant
  - No hard limit to problem size: could use external DDR
- Just one requirement:
  - Execute ***today's*** action, not the ***next*** action
- Which scheduling problem do **you** have?