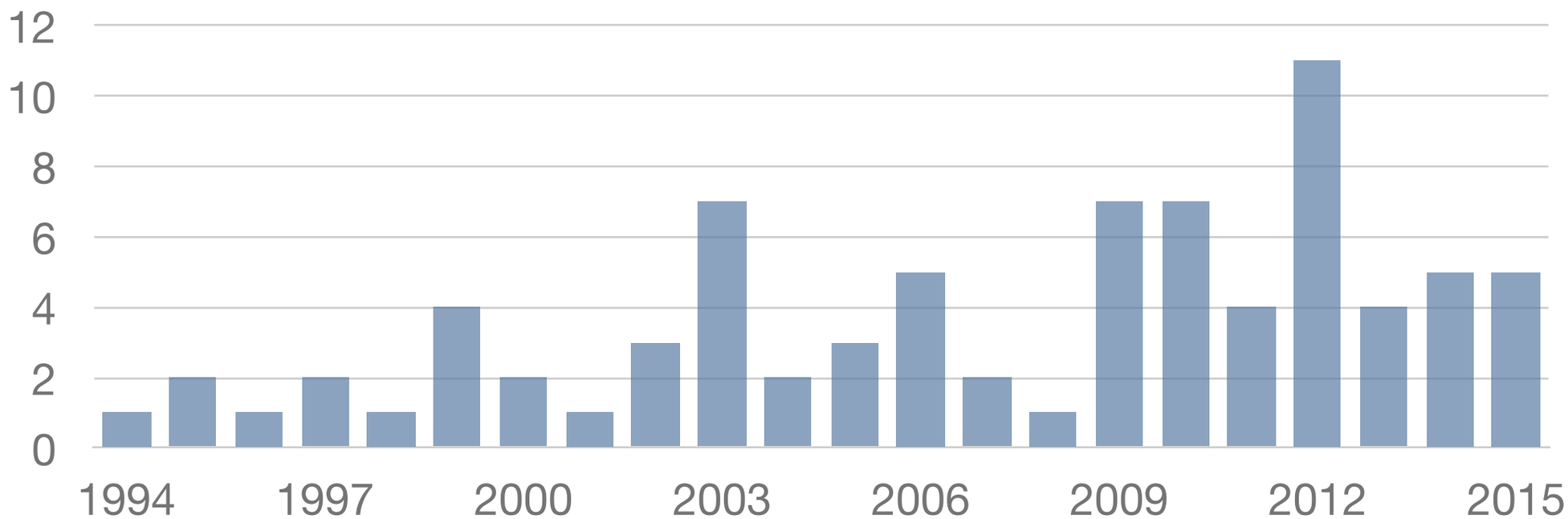
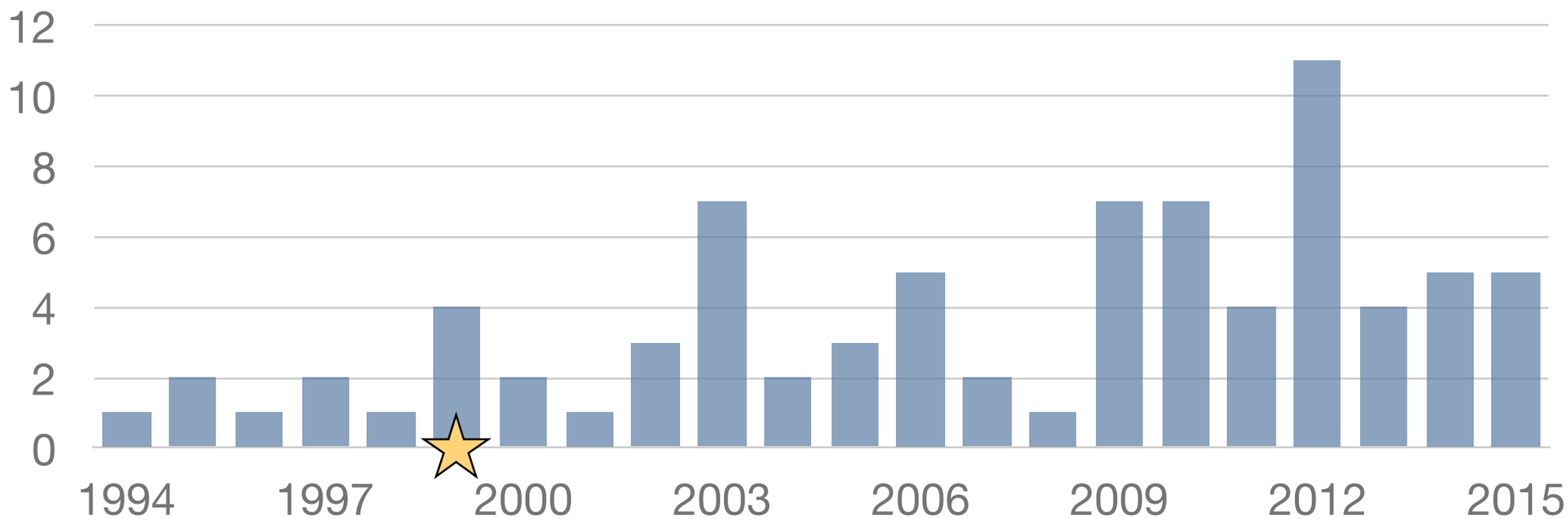
ROSS for PDES Research

Elsa Gonsiorowski
Justin LaPre

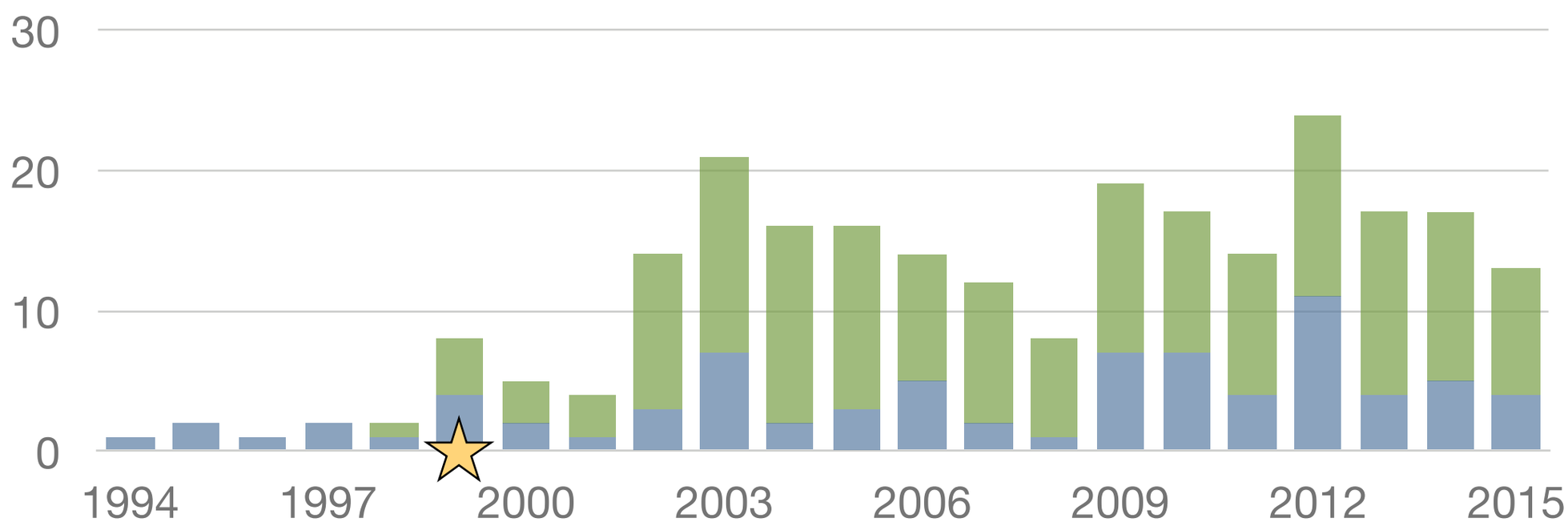
CD Carothers



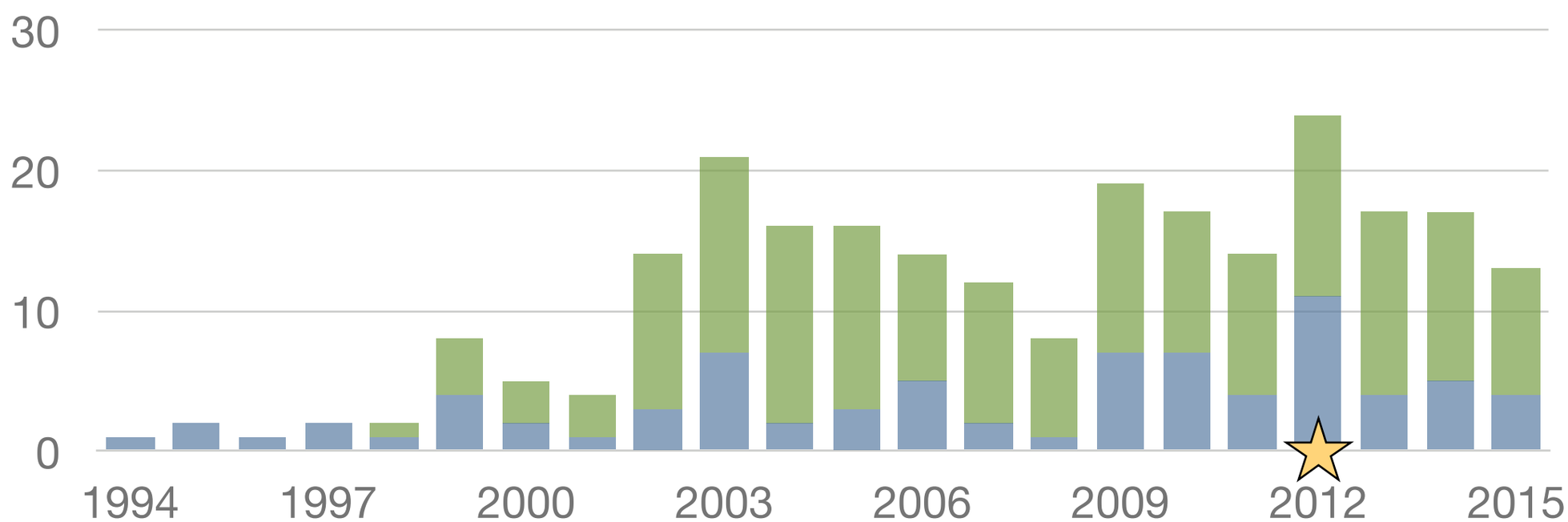
CD Carothers



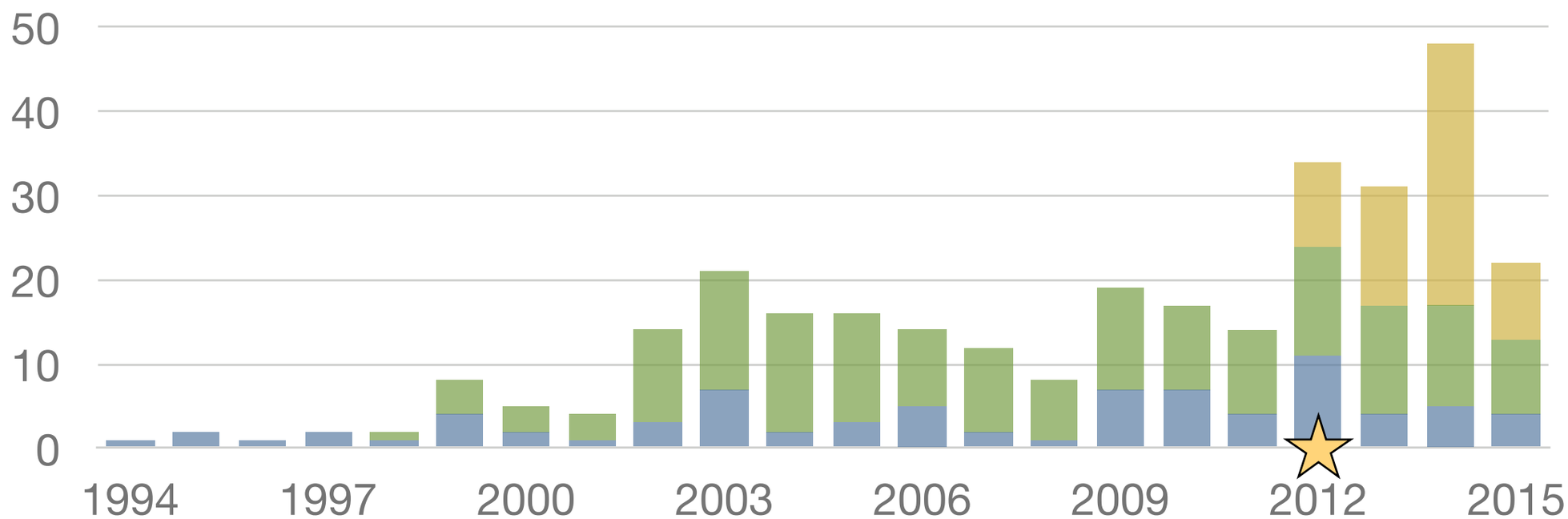
“Efficient Optimistic Parallel Simulations Using Reverse Computation”



“Efficient Optimistic Parallel Simulations Using Reverse Computation”



“On the Role of Burst Buffers in Leadership-Class Storage Systems”



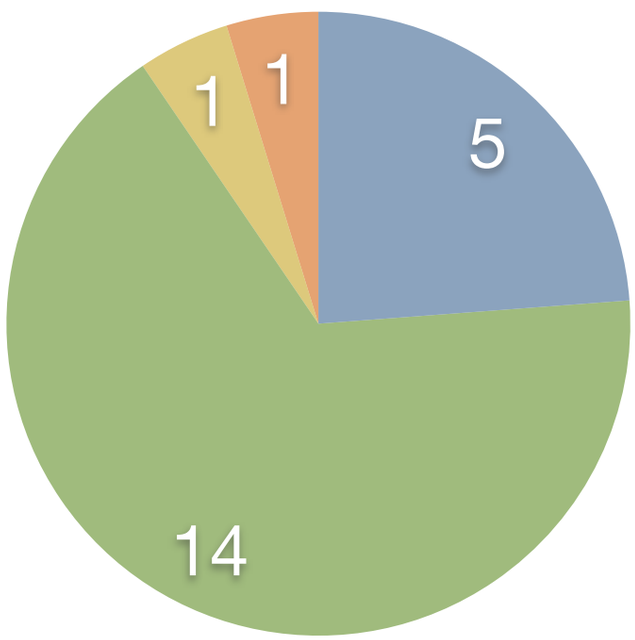
Research vs Software Engineering

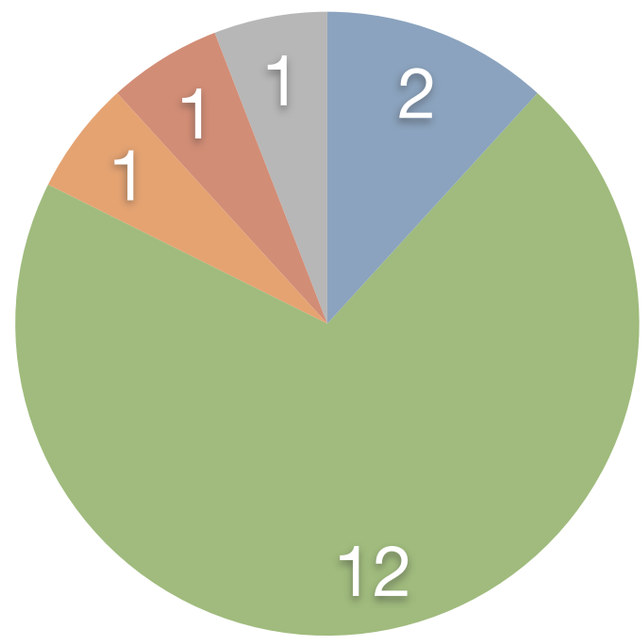
- Doxygen Automated Documentation
- GitHub (and GitHub Issues)
- LP-Printf
- Continuous Integration with Travis
- AVL Tree
- LORAIN
- Delta Encoding

Research vs Software Engineering

- AVL Tree (PADS 2014)
- LORAIN (PADS 2014)
- Delta Encoding (WSC 2015)
- Gates (MASCOTS 2012)
- Automatic Model Generation (PADS 2015)

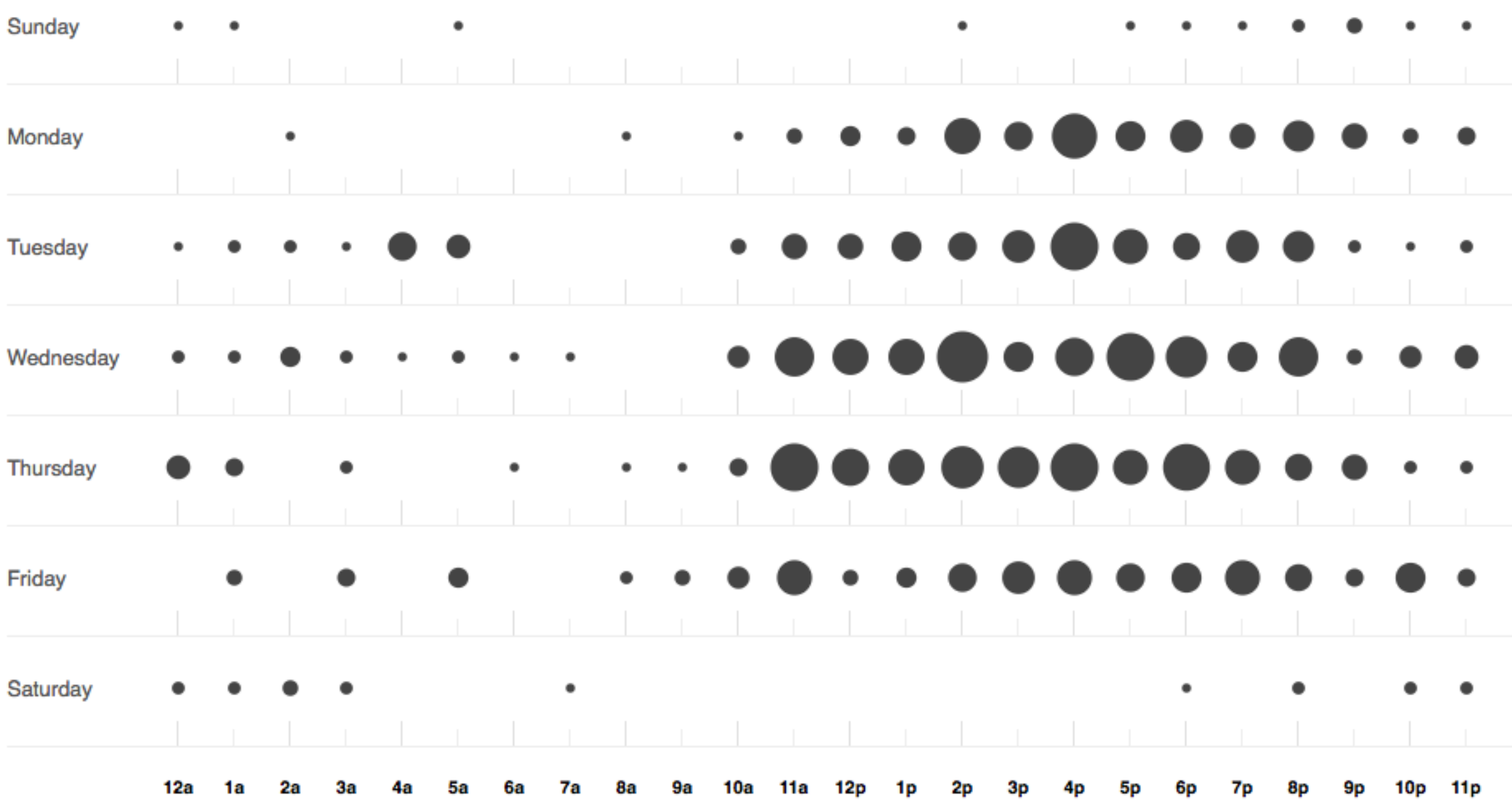
Issues





gonsie
carns
markplagge
JohnPJenkins
laprej
mmubarak

Punch Card

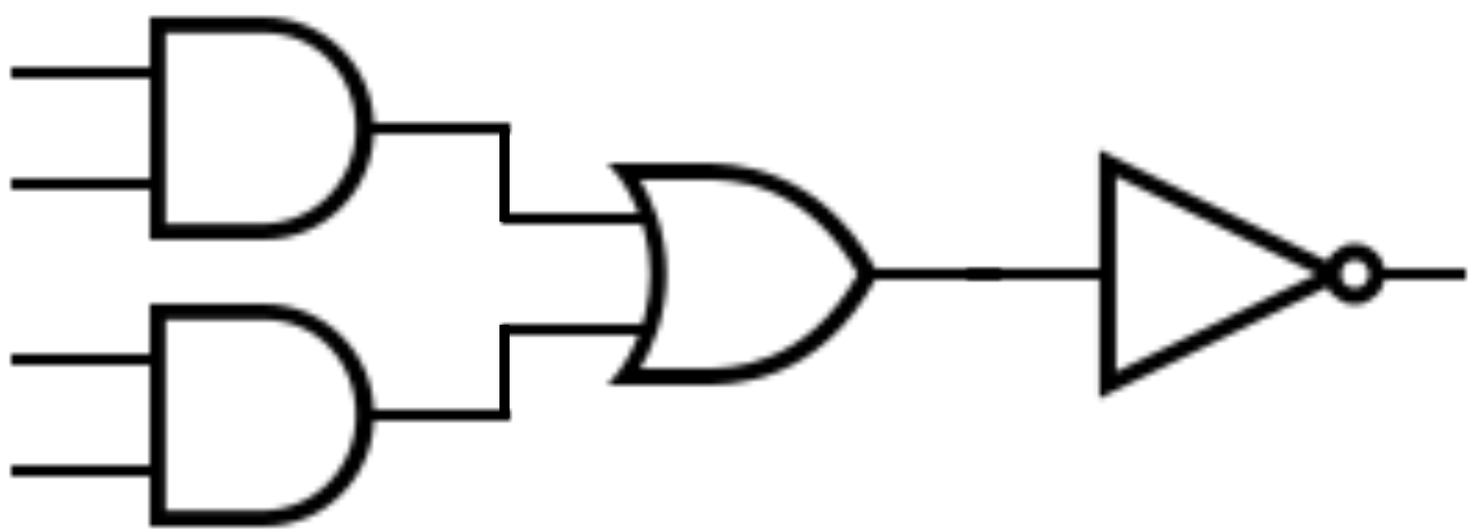

Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
12a 1a 2a 3a 4a 5a 6a 7a 8a 9a 10a 11a 12p 1p 2p 3p 4p 5p 6p 7p 8p 9p 10p 11p

Research

- AVL Tree (PADS 2014)
- LORAIN (PADS 2014)
- Delta Encoding (WSC 2015)
- Gates (MASCOTS 2012)
- Automatic Model Generation (PADS 2015)
- RIO (??)

Gate-Level Circuits Simulation

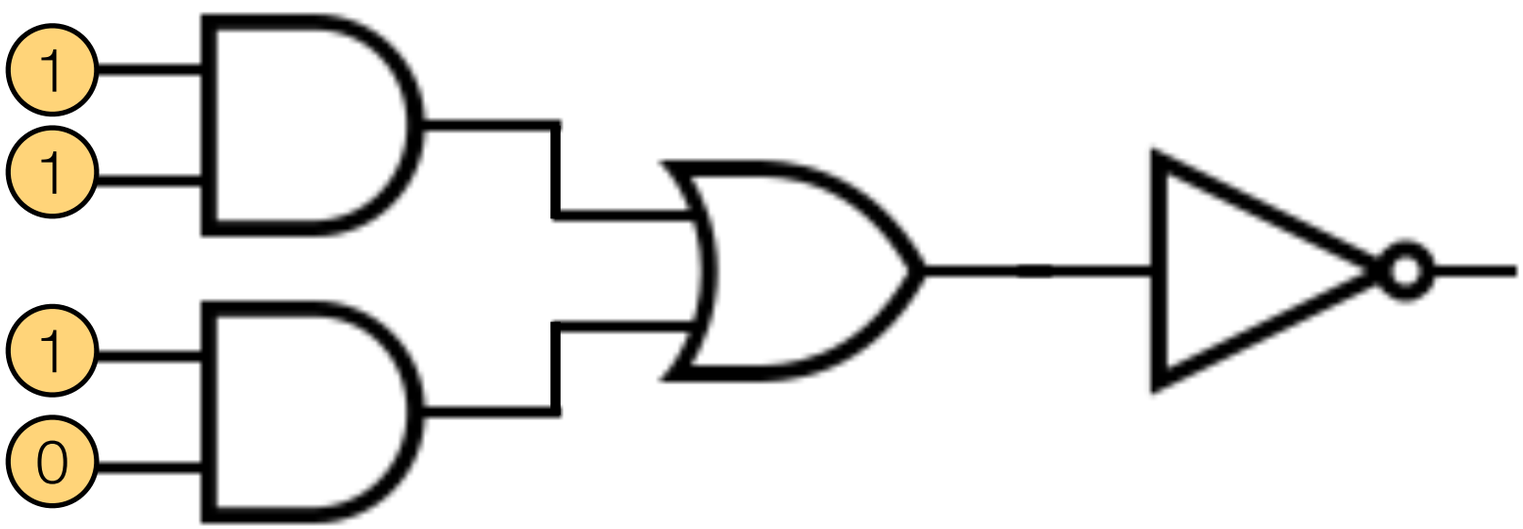
- *Events* are electrical signals
- *Logical processes* (LPs) are the boolean logic gates



Timestamp: 1 2 3 4

Gate-Level Circuits Simulation

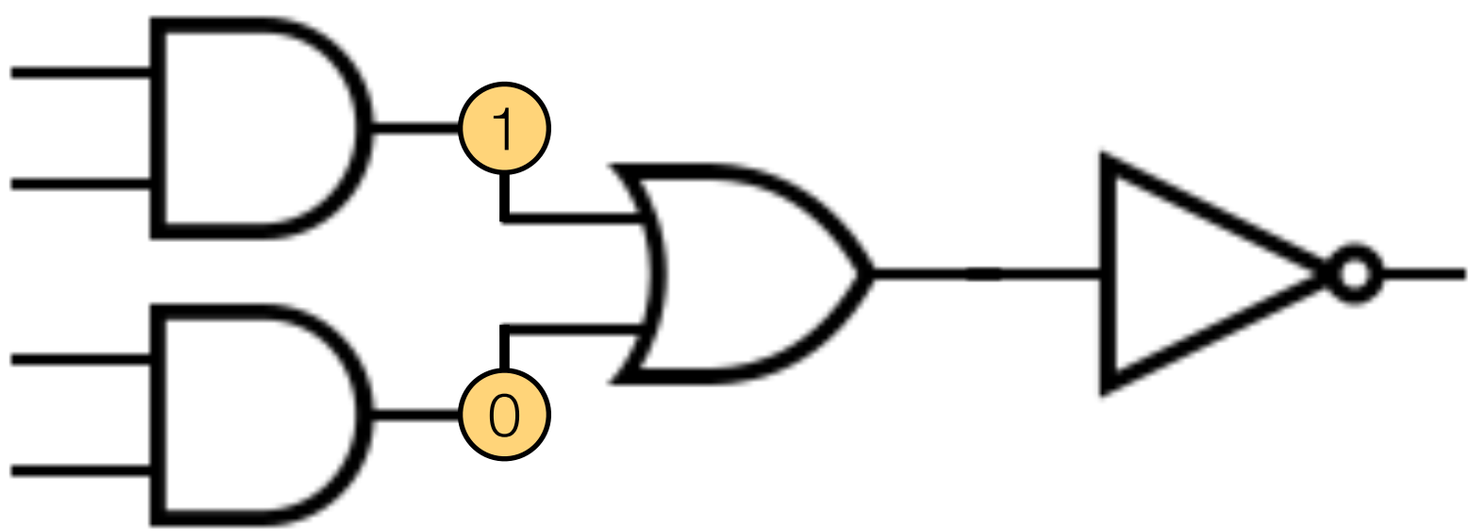
- *Events* are electrical signals
- *Logical processes* (LPs) are the boolean logic gates



Timestamp: 1 2 3 4

Gate-Level Circuits Simulation

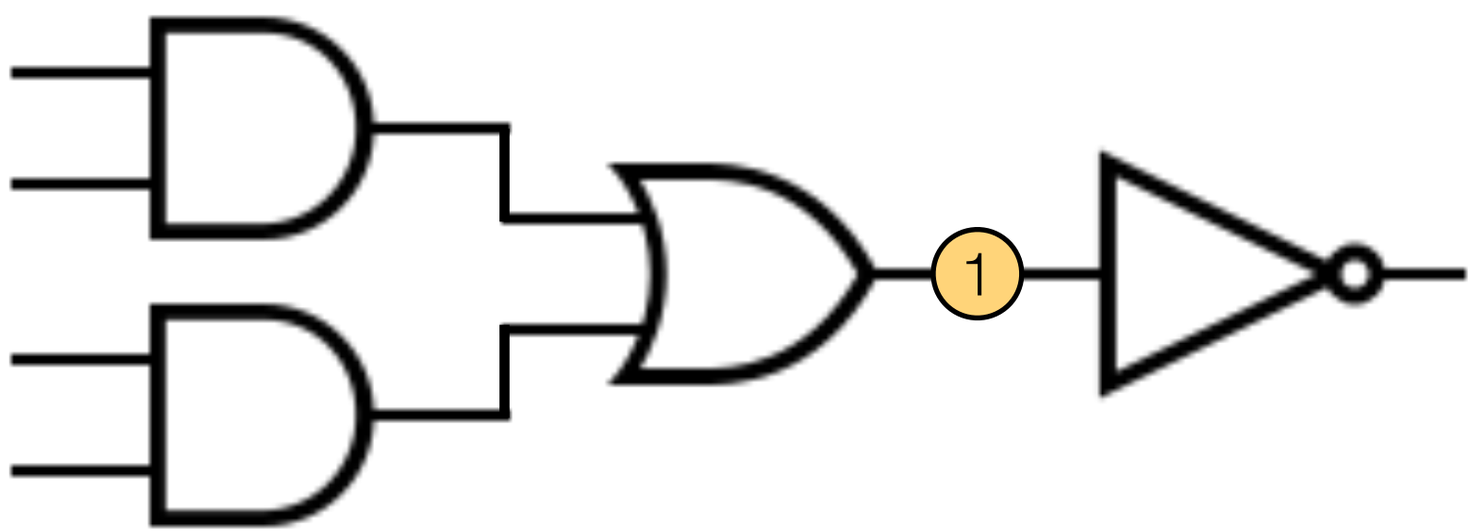
- *Events* are electrical signals
- *Logical processes* (LPs) are the boolean logic gates



Timestamp: 1 2 3 4

Gate-Level Circuits Simulation

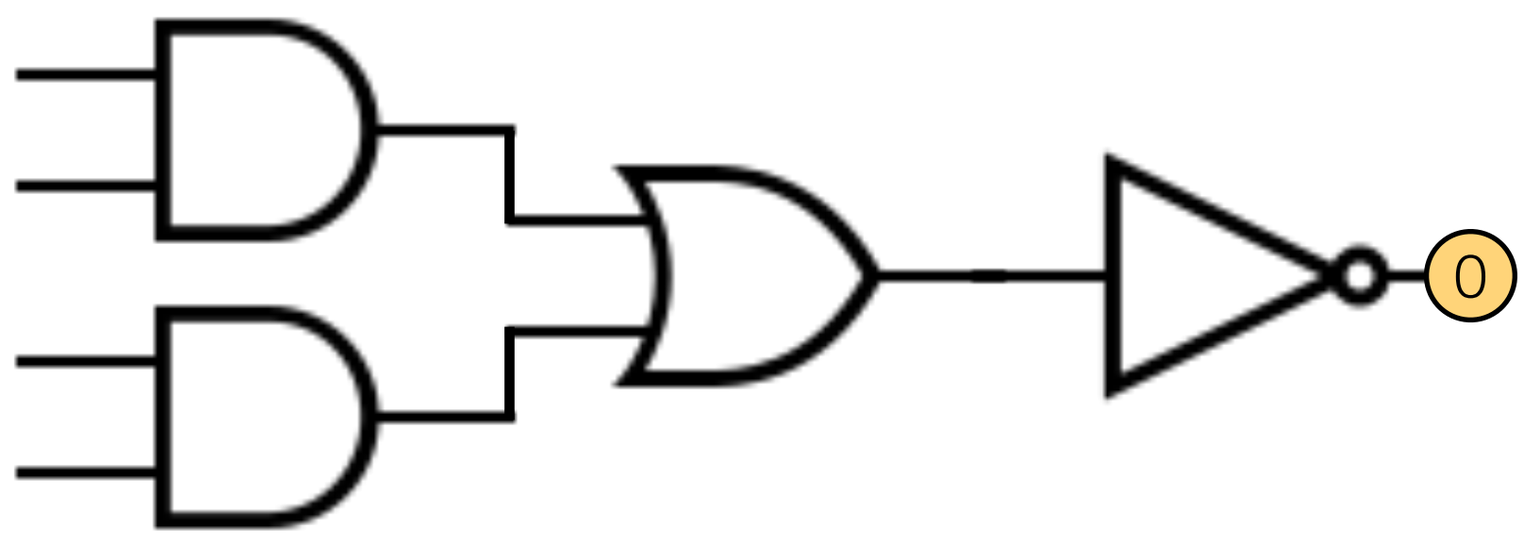
- *Events* are electrical signals
- *Logical processes* (LPs) are the boolean logic gates



Timestamp: 1 2 3 4

Gate-Level Circuits Simulation

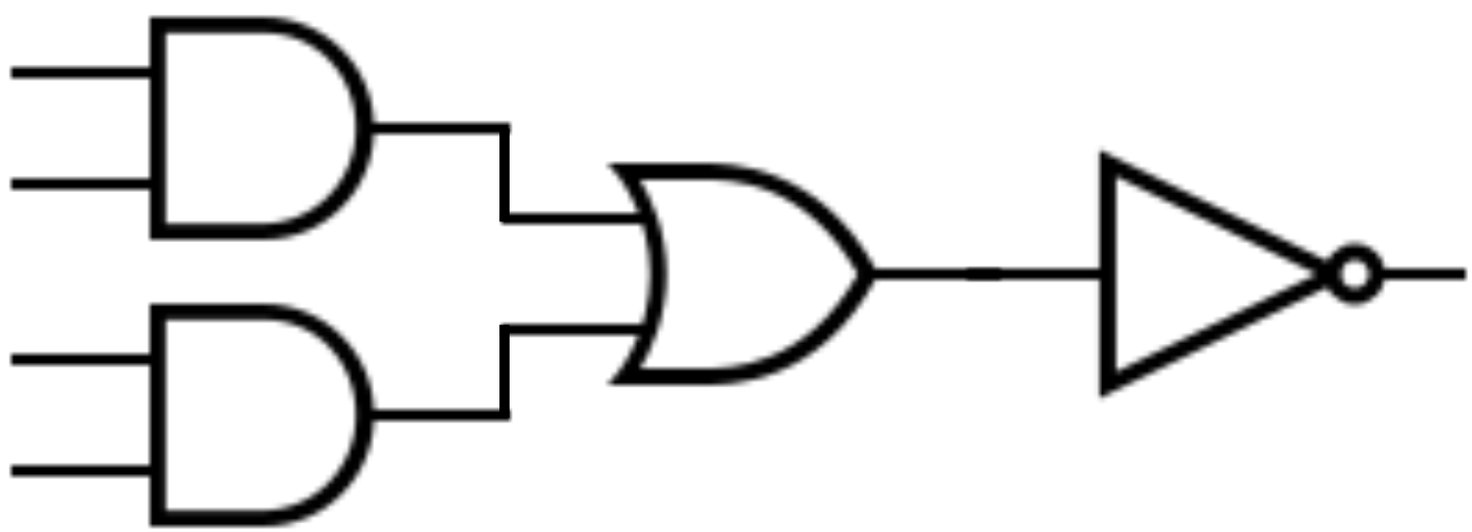
- *Events* are electrical signals
- *Logical processes* (LPs) are the boolean logic gates



Timestamp: 1 2 3 4

Gate-Level Circuits Simulation

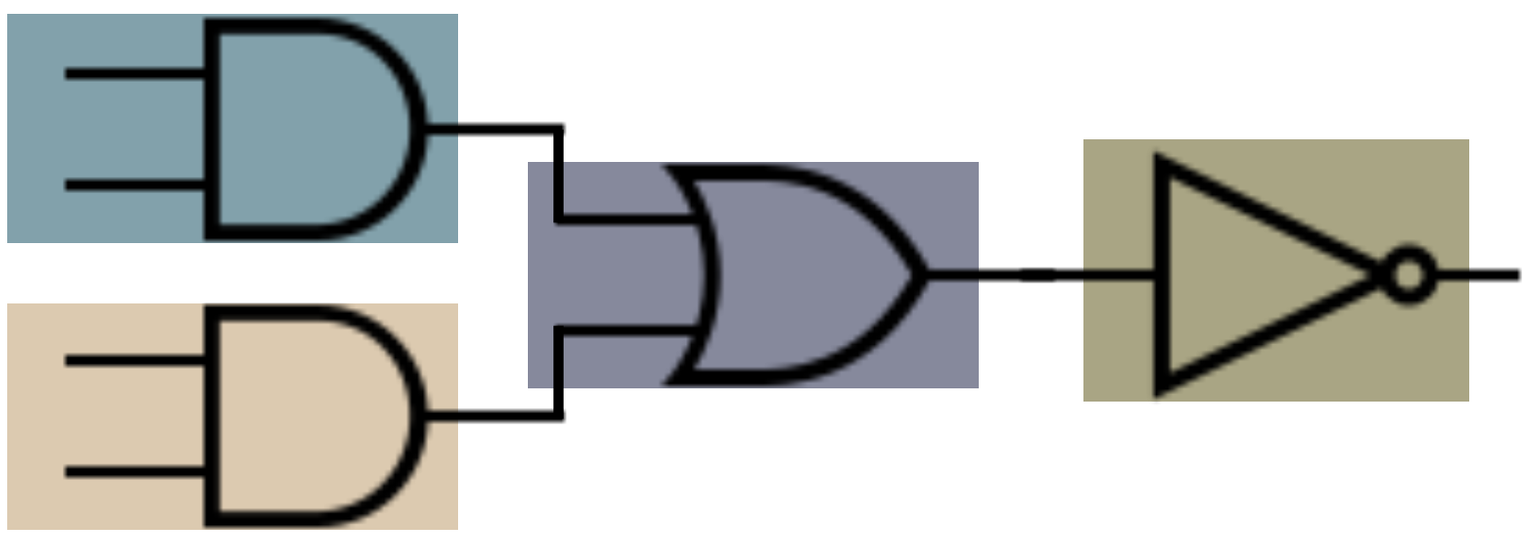
- *Events* are electrical signals
- *Logical processes* (LPs) are the boolean logic gates



Timestamp: 1 2 3 4

PDES: Optimistic Synchronization

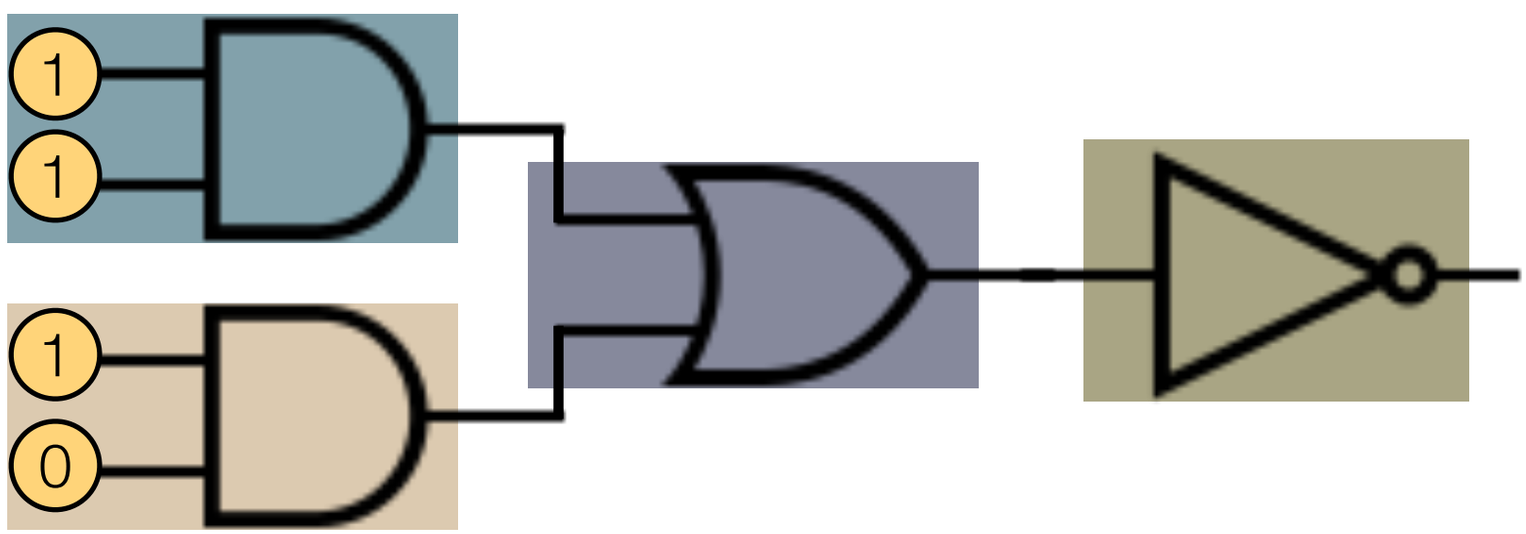
- Time Warp algorithm
  - minimal global time synchronization
  - local recovery when out-of-order event detected
  - Reverse computation and/or state-saving



Timestamp: 1 2 3 4

PDES: Optimistic Synchronization

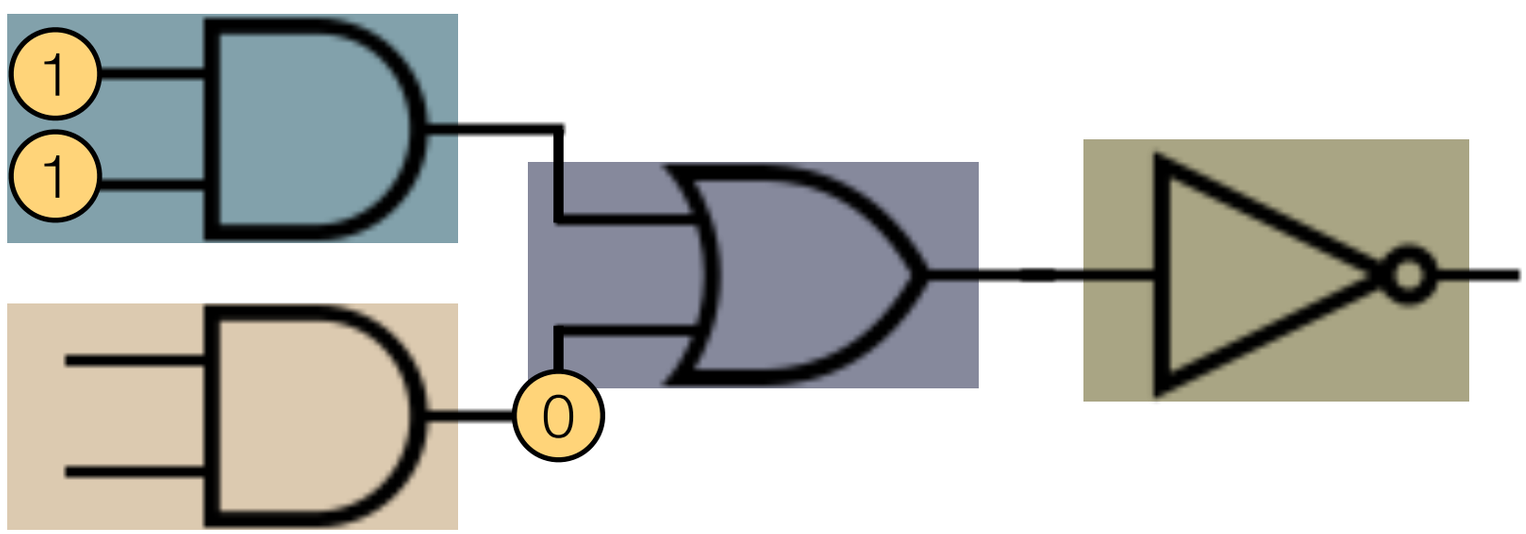
- Time Warp algorithm
  - minimal global time synchronization
  - local recovery when out-of-order event detected
  - Reverse computation and/or state-saving



Timestamp: 1 2 3 4

PDES: Optimistic Synchronization

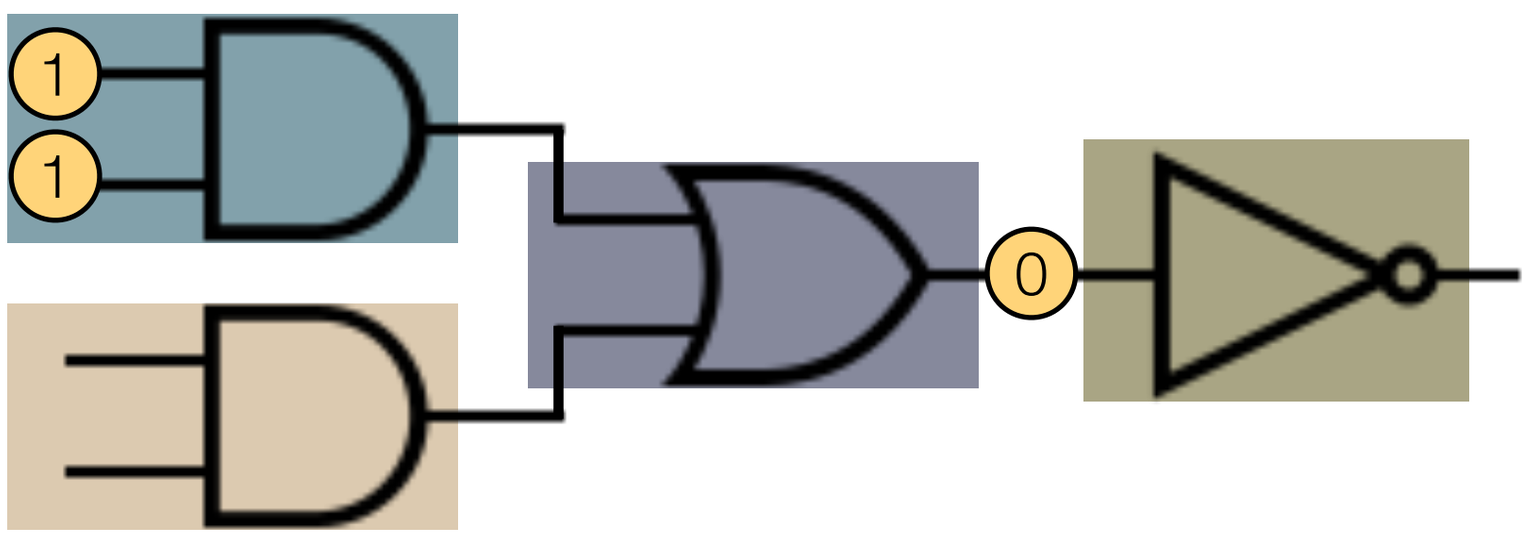
- Time Warp algorithm
  - minimal global time synchronization
  - local recovery when out-of-order event detected
  - Reverse computation and/or state-saving



Timestamp: 1 2 3 4

PDES: Optimistic Synchronization

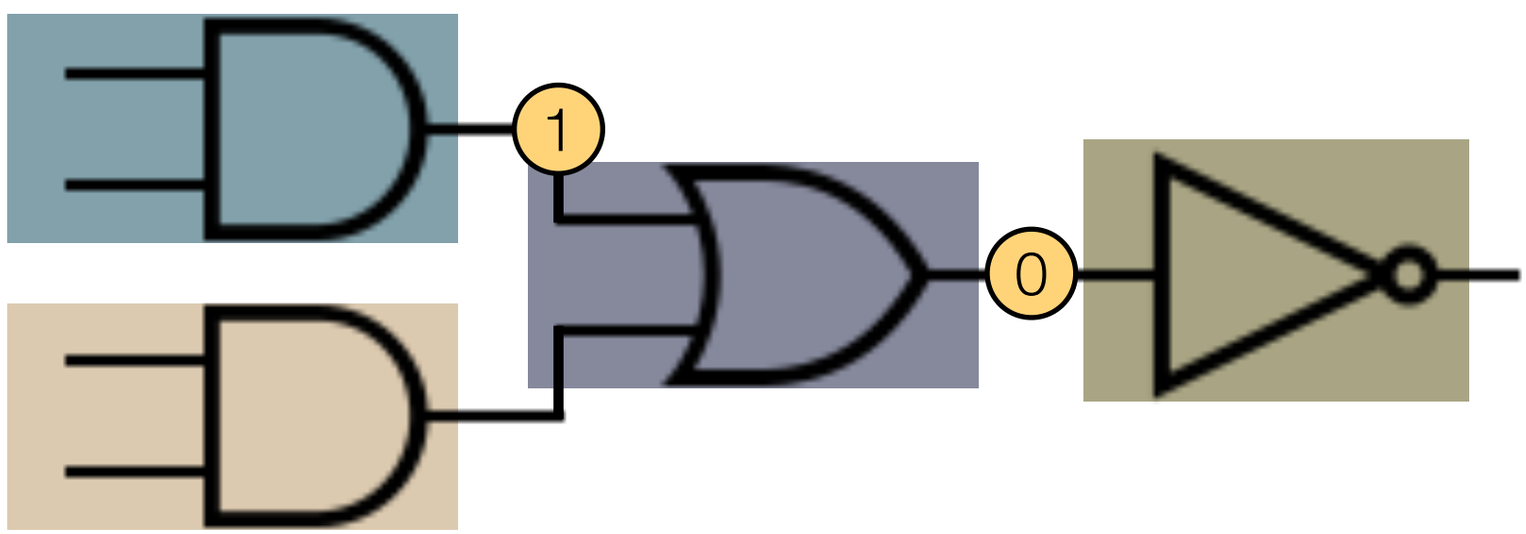
- Time Warp algorithm
  - minimal global time synchronization
  - local recovery when out-of-order event detected
  - Reverse computation and/or state-saving



Timestamp: 1 2 3 4

PDES: Optimistic Synchronization

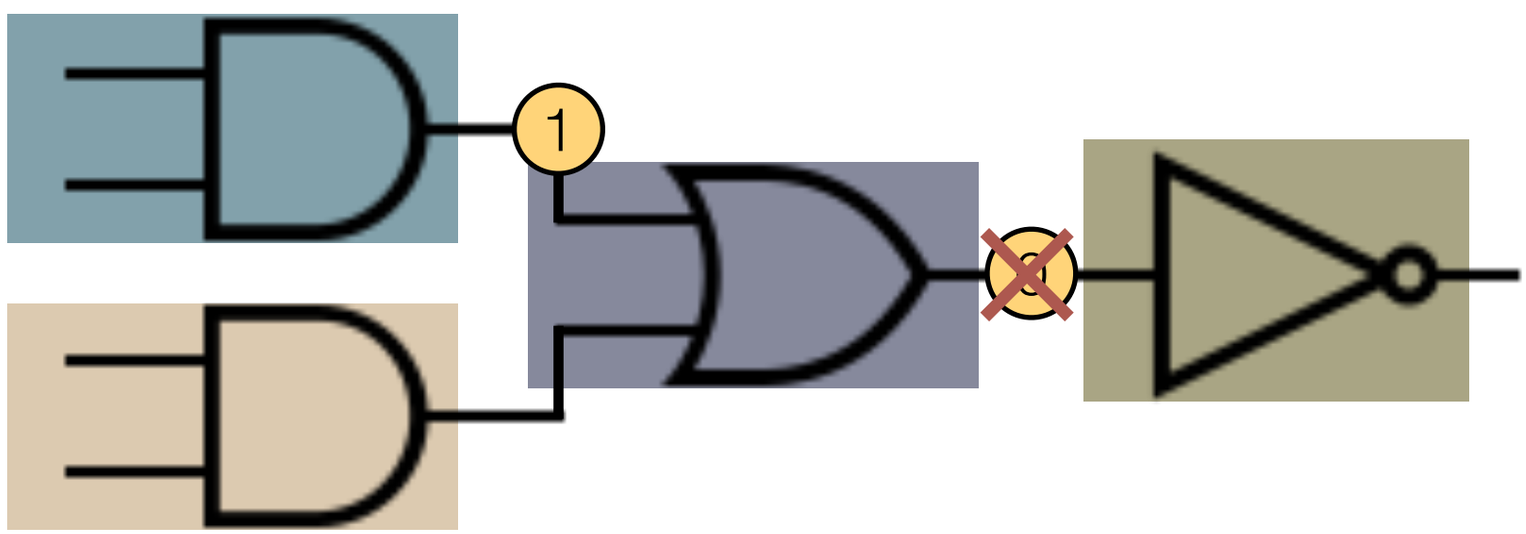
- Time Warp algorithm
  - minimal global time synchronization
  - local recovery when out-of-order event detected
  - Reverse computation and/or state-saving



Timestamp: 1 2 3 4

PDES: Optimistic Synchronization

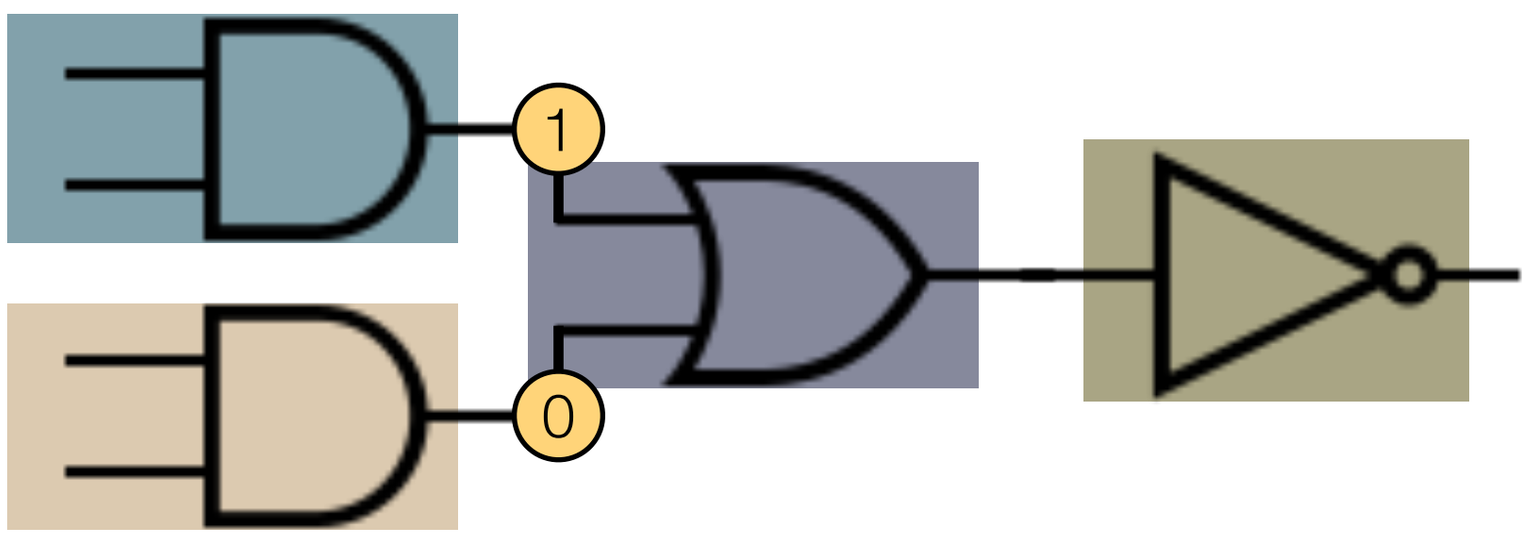
- Time Warp algorithm
  - minimal global time synchronization
  - local recovery when out-of-order event detected
  - Reverse computation and/or state-saving



Timestamp: 1 2 3 4

PDES: Optimistic Synchronization

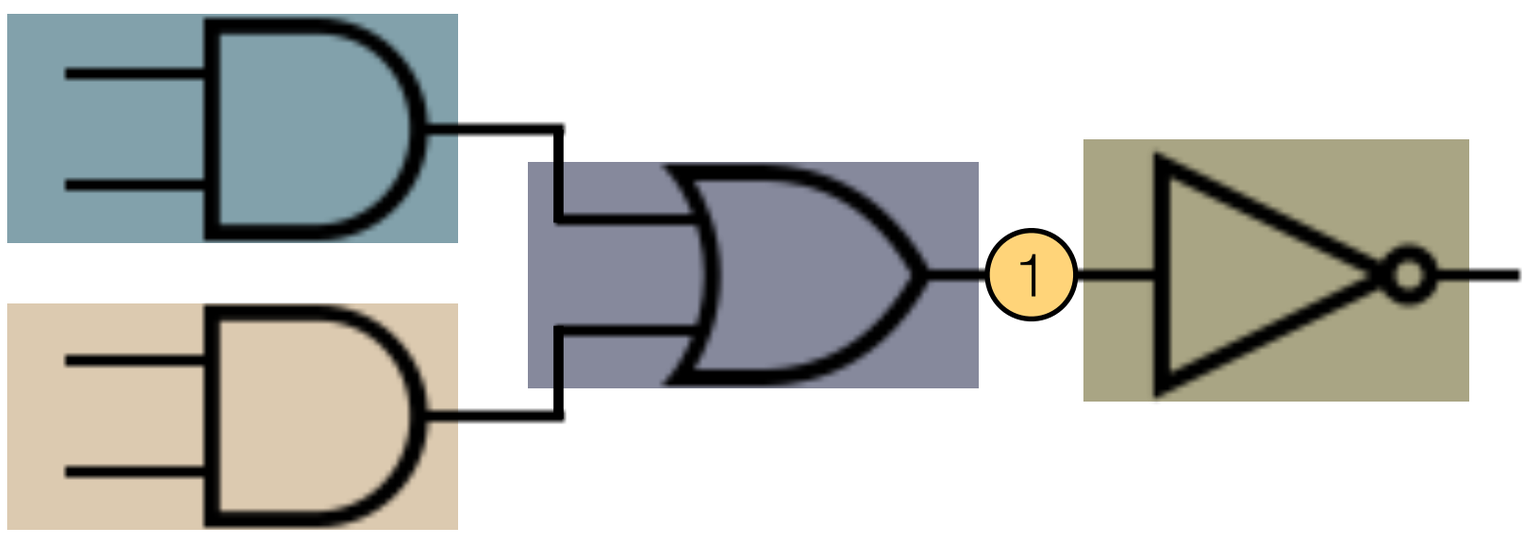
- Time Warp algorithm
  - minimal global time synchronization
  - local recovery when out-of-order event detected
  - Reverse computation and/or state-saving



Timestamp: 1 2 3 4

PDES: Optimistic Synchronization

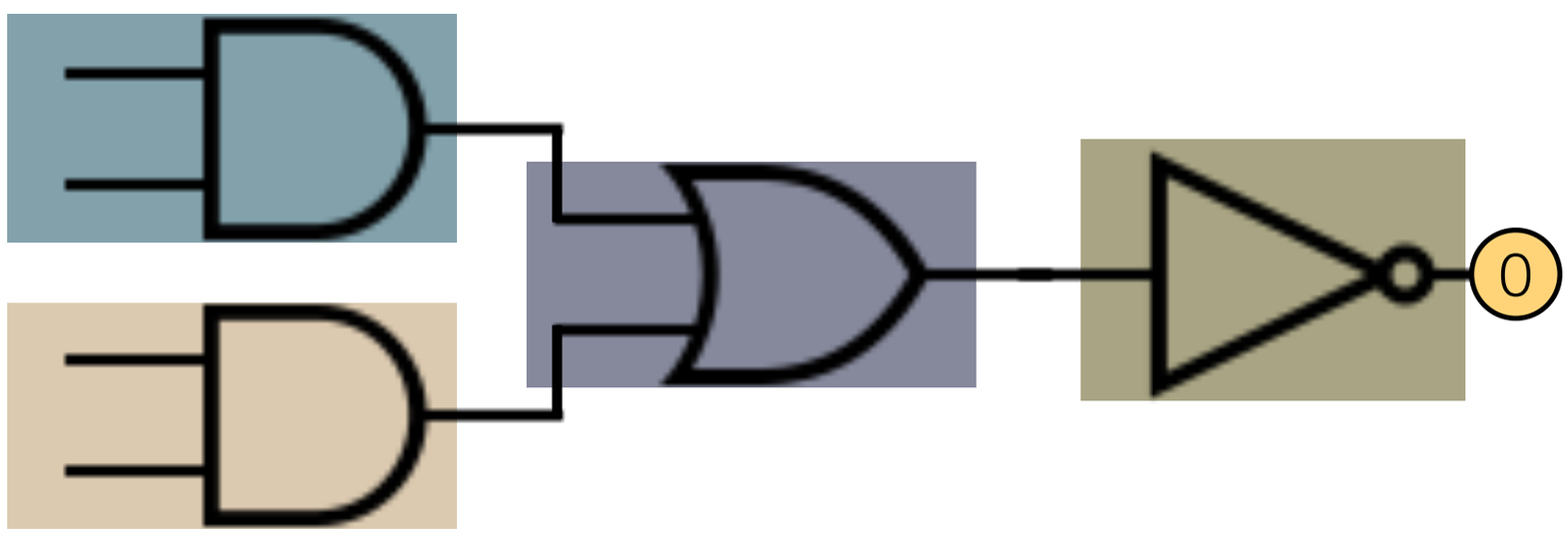
- Time Warp algorithm
  - minimal global time synchronization
  - local recovery when out-of-order event detected
  - Reverse computation and/or state-saving



Timestamp: 1 2 3 4

PDES: Optimistic Synchronization

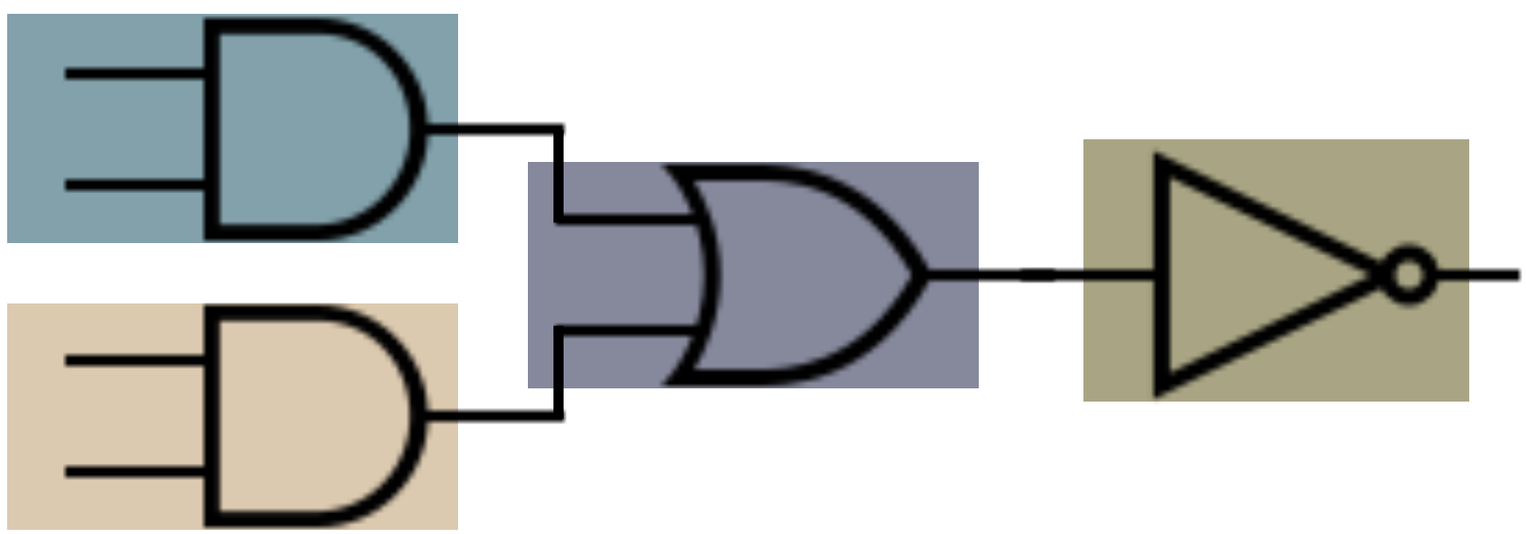
- Time Warp algorithm
  - minimal global time synchronization
  - local recovery when out-of-order event detected
  - Reverse computation and/or state-saving



Timestamp: 1 2 3 4

PDES: Optimistic Synchronization

- Time Warp algorithm
  - minimal global time synchronization
  - local recovery when out-of-order event detected
  - Reverse computation and/or state-saving



Timestamp: 1 2 3 4

PDES Research

- How can we automate event rollback?
- Incremental state-saving with delta encoding
- Automatic reverse computation with LORAIN

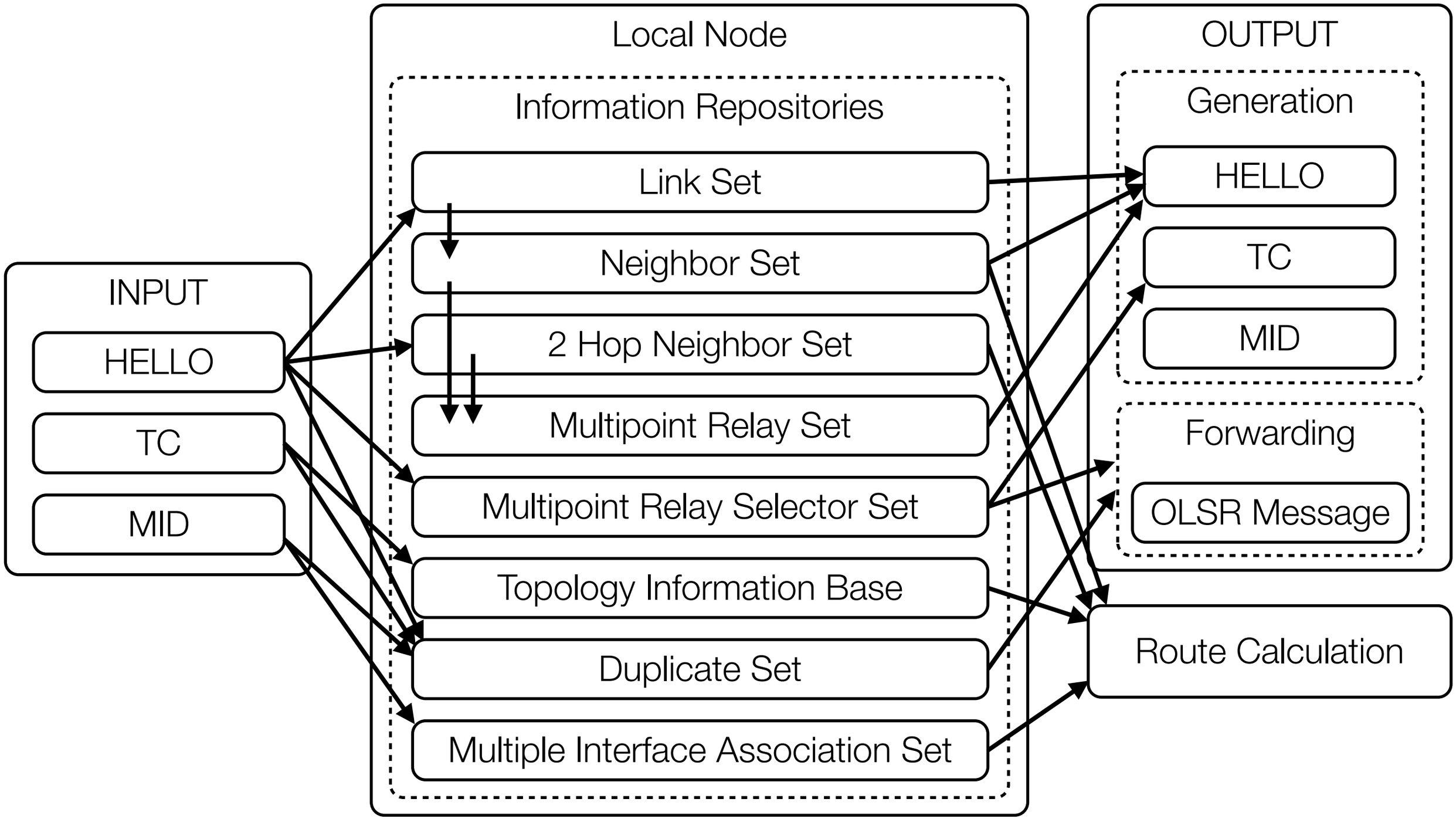
Delta Encoding

- **Motivation: not all event handlers are reversible, e.g., while loop containing if statements**
- We have to fall back to state-saving, but consumes too much memory!
- Source control systems typically store changes as *deltas*
  - Often a substantial win when changes are small
- PDES state changes typically small
- Deltas on sizable models will likely have many zeroes, i.e., things are the same
- Excellent opportunity for compression!

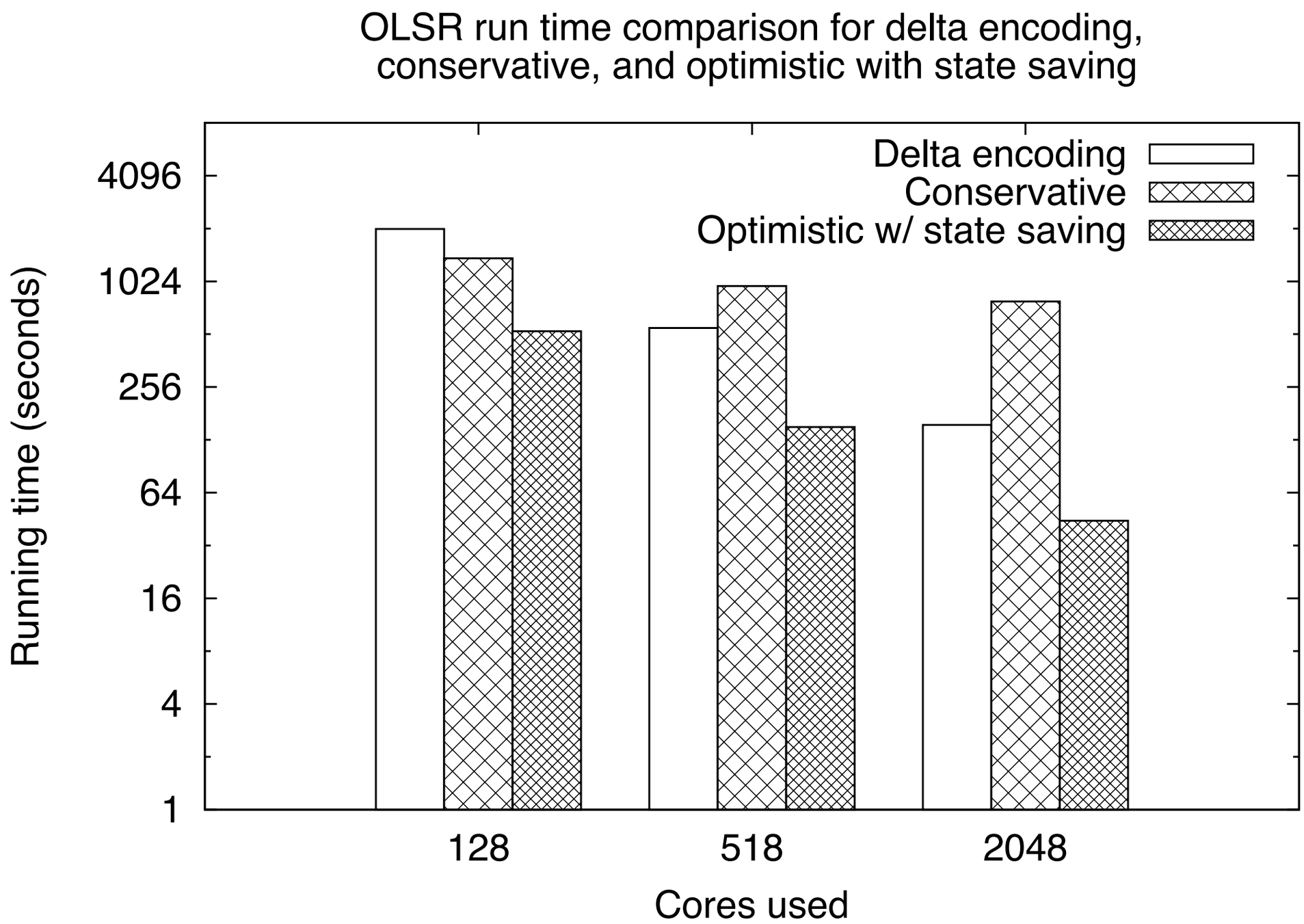
Approach

- In the main event loop in ROSS:
  - memcpy() LP state into a buffer $state_{before}$
  - run event
  - Now we have $state_{after}$
  - Call delta_diff function on $state_{before}$ and $state_{after}$
  - Compress diff with LZ4, save with event

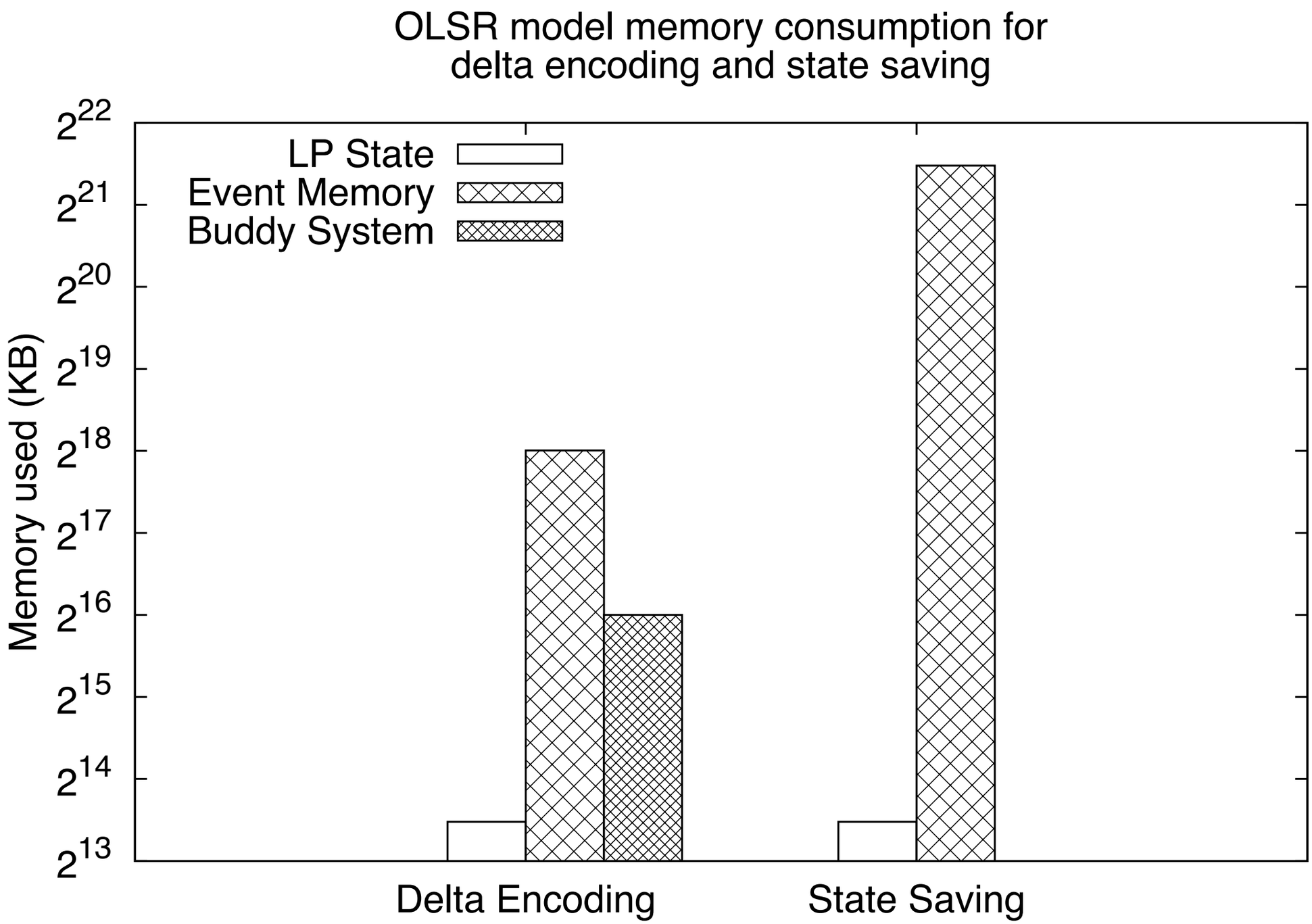
Example: OLSR Protocol



OLSR Performance for Various Approaches



Delta Encoding Uses Far Less Memory!



Motivation Behind LORAIN

- Writing reverse code is hard!
- Example: swapping into messages is common!

```
// Forward
if (msg->val == 1) {
    SWAP(lp->val, msg->val);
}

// Reverse
if (msg->val == 1) {
    SWAP(lp->val, msg->val);
}
```

Looks “obviously” correct forward and backward, but it’s not.

```
// Forward
if (msg->val == 1) {
    SWAP(lp->val, msg->val);
}

// Reverse
if (lp->val == 1) {
    SWAP(lp->val, msg->val);
}
```

Hard error to spot!

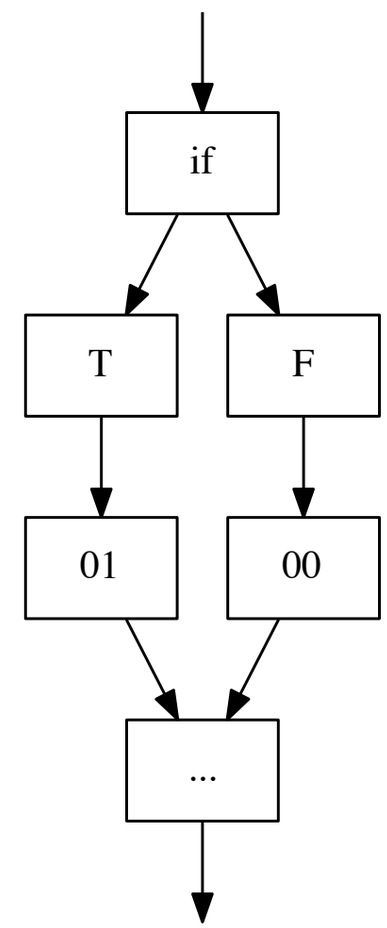
What Does LLVM IR Look Like?

```
void test_if(void)
{
    if (test_if_x_condition) {
        test_if_x = test_if_x + 1;
    }
}
```

```
define void @test_if() nounwind uwtable ssp {
  %1 = load i32* @test_if_x_condition, align 4
  %2 = icmp ne i32 %1, 0
  br i1 %2, label %3, label %6

; <label>:3    ; preds = %0
  %4 = load i32* @test_if_x, align 4
  %5 = add nsw i32 %4, 1
  store i32 %5, i32* @test_if_x, align 4
  br label %6

; <label>:6    ; preds = %3, %0
  ret void
}
```



3 Steps

- Augment the appropriate “message” data structure to support swap operations
  - Catch “destructive” operations e.g., `x = 5`
- Instrument / analyze the forward event handler
  - Mark non-important `store` instructions with metadata
- Clone and invert the forward event handler

Continuing Research
RIO: Checkpointing API For ROSS

- checkpoint.readme.txt
- checkpoint.metadata.mh
- checkpoint.data-#

Using RIO

- Compile in with CMake `USE_RIO` option
- `--io-parts=n` and `--io-files=n` flags or function call
- `io_lptype` struct
- Mapping extension
- Implement: Model size function
- Implement: Serialize and deserialize functions

YOUR PAPER HERE

T. Liu, N. Wolfe, C. D. Carothers, Wei Ji, and X. George Xu,``Optimizing the Monte Carlo neutron cross-section construction code, XSBench, to MIC and GPU platforms'', In Proceedings the ANS MC2015 - Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Methods, Nashville, TN, April 19-23, 2015.

N. Wolfe, C. D. Carothers, T. Liu and G. Xu, ``Concurrent CPU, GPU and MIC Execution Algorithms for Archer Monte Carlo Code Involving Photon and Neutron Radiation Transport Problems'', In Proceedings the ANS MC2015 - Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Methods, Nashville, TN, April 19-23, 2015.

M. Mubarak, C. D. Carothers, P. Carns and R. B. Ross, ``Using Massively Parallel Simulation for MPI Collective Communication Modeling in Extreme-Scale Networks'', In Proceedings of the 2014 Winter Simulation Conference, Savannah GA, December, 2014.

S. Snyder, P. Carns, R. B. Ross, J. Jenkins, K. Harms, M. Mubarak and C. D Carothers, ``A Case for Epidemic Fault Detection and Group Membership in HPC Storage Systems'', In Proceedings of the 5th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS 2015) as part of Supercomputing (SC'14), New Orleans, LA, November 2014.