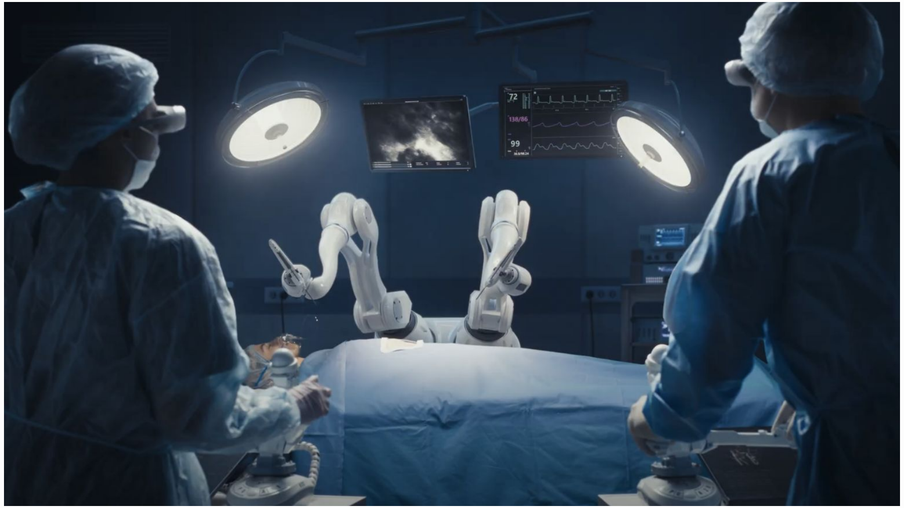
Faster, Exact, More General Response-time Analysis for NVIDIA Holoscan Applications

Philip Schowitz, Shubhaankar Sharma, Siddharth Balodi, Bruce Shepherd, Arpan Gujarati
The University of British Columbia
Soham Sinha
NVIDIA

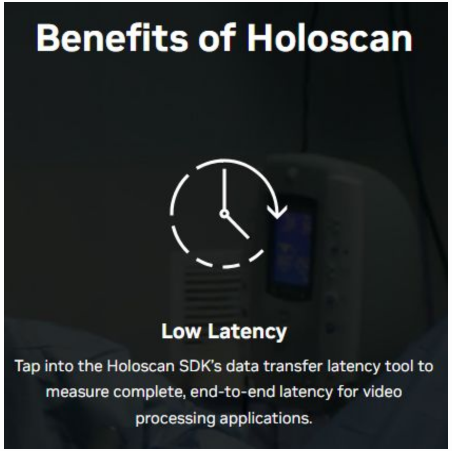
Edge Computing

- AI is fueling resource-intensive applications on the edge
- Embedded platforms become more complex
  - Harder to develop apps



Frameworks

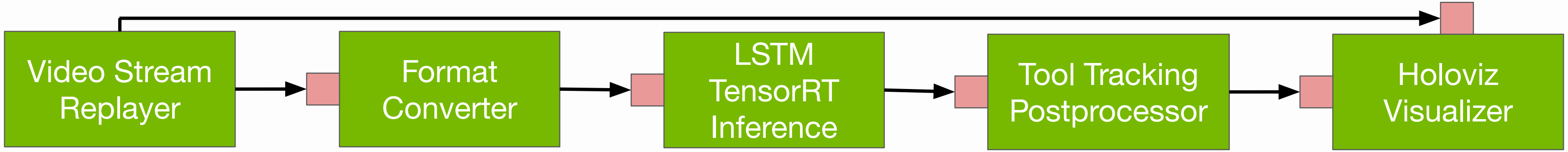
- Frameworks like **NVIDIA Holoscan** streamline development
- Lacks hard latency guarantees
  - Potential for framework-level analysis



Why care about latency analysis?

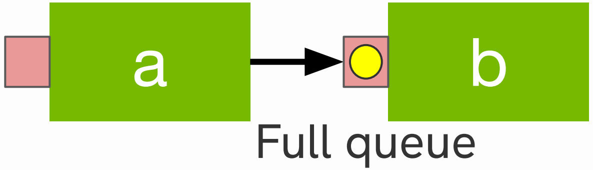
- Developers lack knowledge about framework timing properties but want low latency
  - Medical imaging, robotics
  - In the future, robotic surgery
- Create static analysis to allow informed decisions
  - E.g., what will be the timing effect of adding this node?
  - Useful during both design and development stages

Holoscan Basics



- Apps are represented as directed acyclic graphs
- App can process multiple inputs in parallel
  - Each node only processes one item at a time
  - Assume we can run all nodes at once
    - No node-to-core scheduling

Challenges



- Holoscan applications use a unique DAG model with downstream conditions
  - Node cannot begin execution if downstream queue is full

Challenges

- Holoscan applications use a unique DAG model with downstream conditions
    - Node cannot begin execution if downstream queue is full

Challenges

- Holoscan applications use a unique DAG model with downstream conditions
  - Node cannot begin execution if downstream queue is full

Challenges

- Holoscan applications use a unique DAG model with downstream conditions
  - Node cannot begin execution if downstream queue is full
  - Implements static backpressure, maintains internal consistency by preventing queue overflow

Challenges

- Holoscan applications use a unique DAG model with downstream conditions
  - Node cannot begin execution if downstream queue is full
  - Implements static backpressure, maintains internal consistency by preventing queue overflow
- This design creates timing anomalies
  - Lower node execution time → higher global response time

Prior Work

- We created a response time analysis for this model
  - Safe upper bound, scalable

Schowitz et al., RTSS 2024, “Response-Time Analysis of a Soft Real-time NVIDIA Holoscan Application”

Prior Work

- We created a response time analysis for this model
  - Safe upper bound, scalable
- However...
  - Pessimistic for some graphs
  - Static execution time, queue size must equal 1
  - Unclear connection to SDFG literature

Schowitz et al., RTSS 2024, “Response-Time Analysis of a Soft Real-time NVIDIA Holoscan Application”

Prior Work

Metric	Prior RTA
Runtime	51 μs
Pessimism	20.5%
Safe	✓
Exact	✗
Variable Execution Time	✗
Variable Queue Size	✗

Specific 9-node graph with pessimism

Contributions

Metric	Prior RTA	Synchronous Dataflow
Runtime	$51\ \mu s$	$6055\ s$
Pessimism	20.5%	0%
Safe	✓	✓
Exact	✗	✓
Variable Execution Time	✗	✓
Variable Queue Size	✗	✓

Specific 9-node graph with pessimism

Contributions

Metric	Prior RTA	Synchronous Dataflow	This work
Runtime	$51\,\mu s$	$6055\,s$	$136\,ms$
Pessimism	20.5%	0%	0%
Safe	✓	✓	✓
Exact	✗	✓	✓
Variable Execution Time	✗	✓	✓
Variable Queue Size	✗	✓	✓

Specific 9-node graph with pessimism

Response-Time Analysis

Problem Statement

- What is the largest possible difference in time between source (a) starting and sink (c) starting?
  - In some iteration x, the difference is: $\mathbf{s(c_x) - s(a_x)}$
  - Can trivially add back sink's exec to get the WCRT

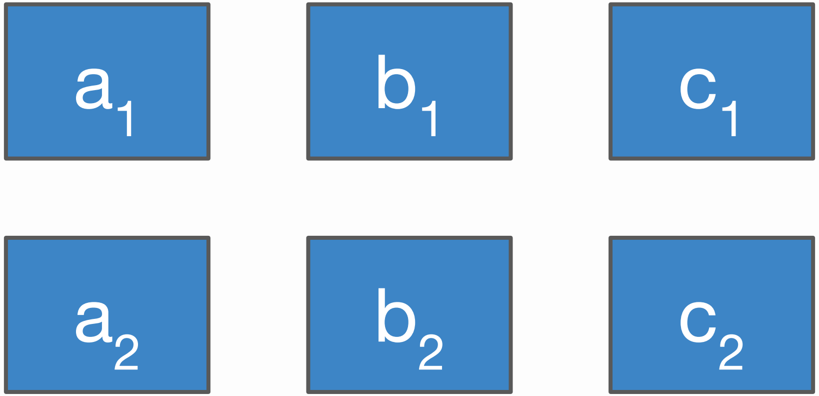
Trace Graph

First iteration
or input

Trace Graph

First iteration or input

a_1 / a_2 may have different execution times, b_1 / b_2, etc

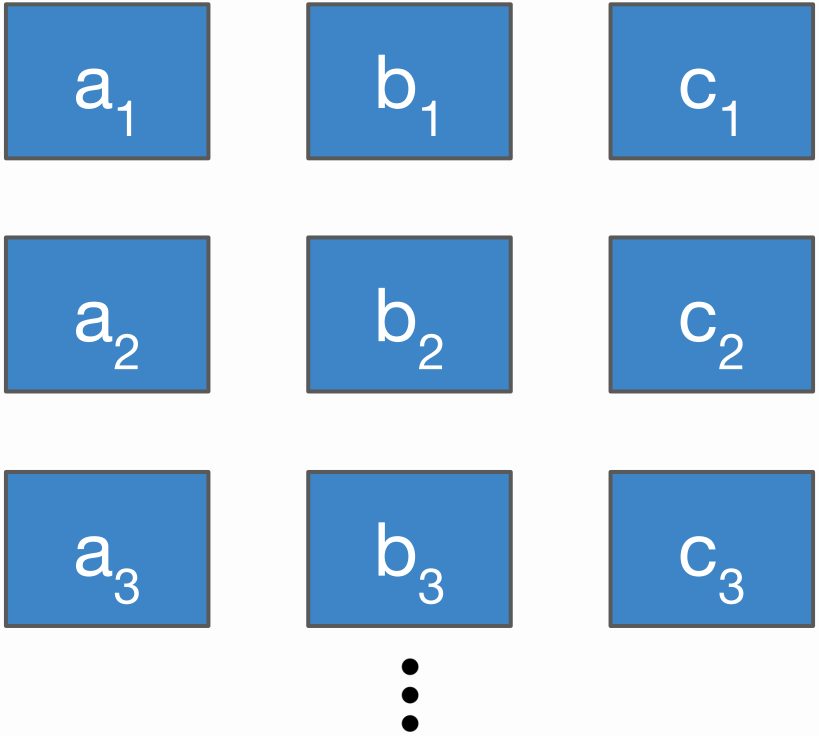


Trace Graph

First iteration or input

a_1 / a_2 may have different execution times, b_1 / b_2, etc

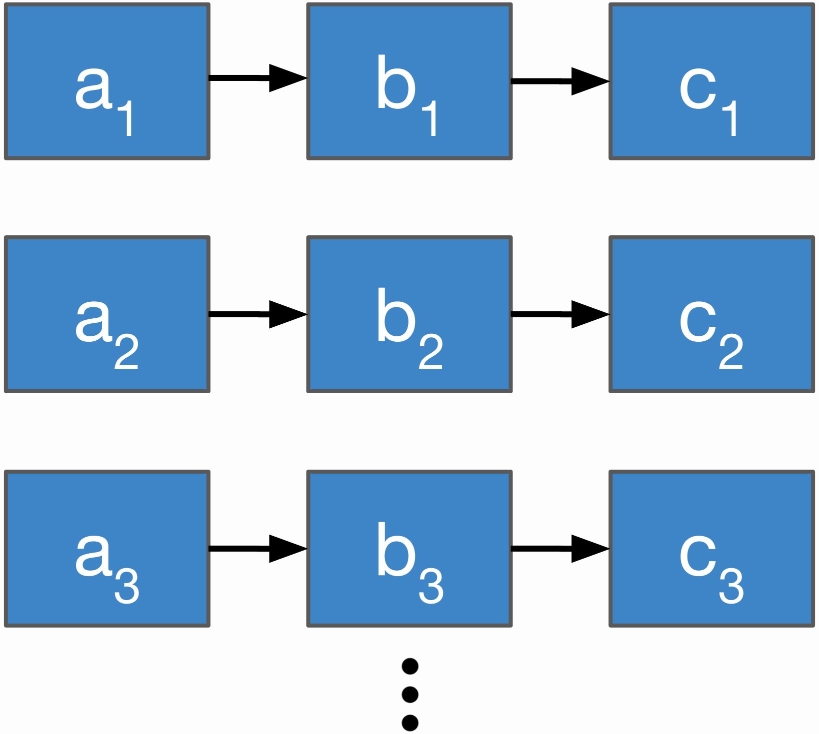
Infinite graph!



Trace Graph

Data-dependency edges

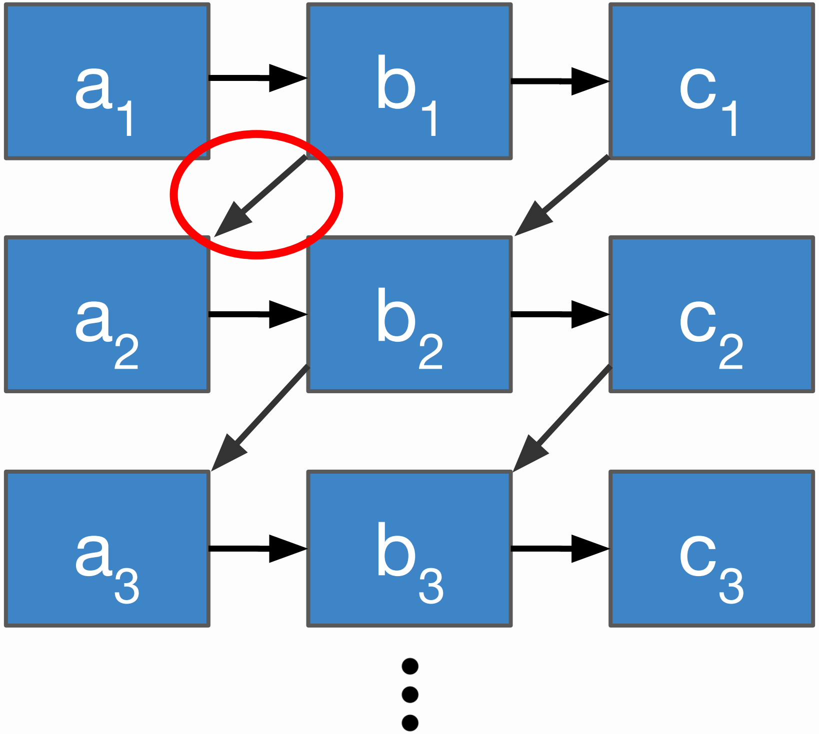
Cost = node execution time (finish-to-start)



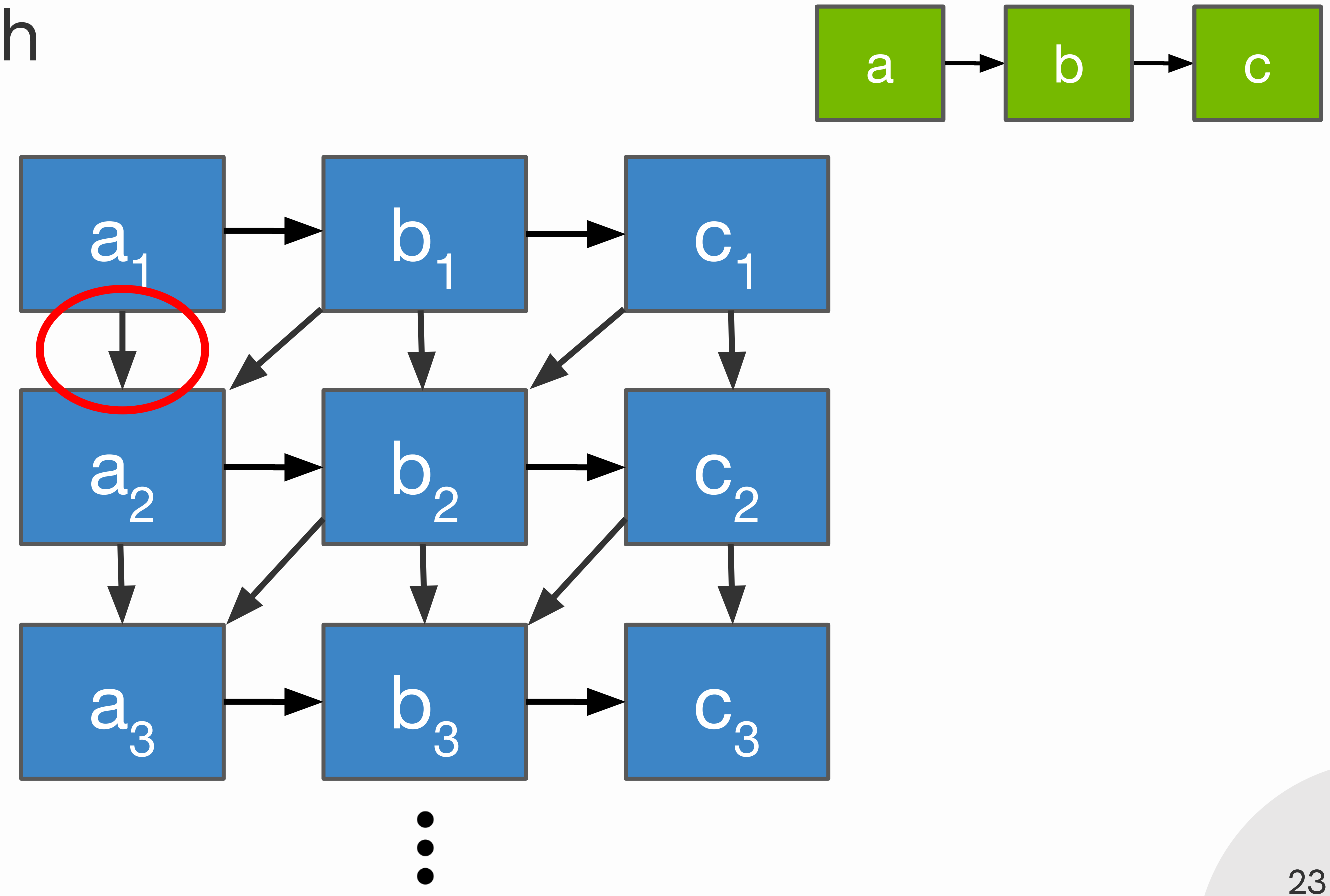
Trace Graph

Downstream blocking edges (backpressure)

Cost = 0 (start-to-start)



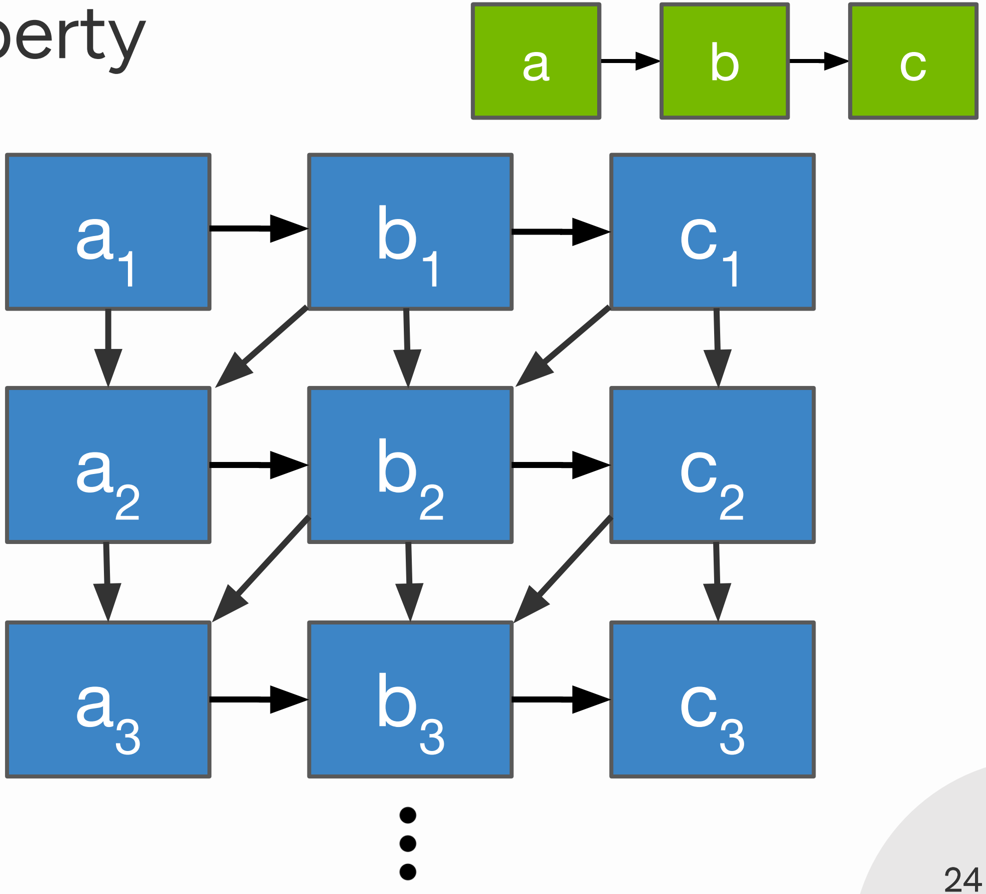
Trace Graph



Sequential-execution edges

Cost = node execution time (finish-to-start)

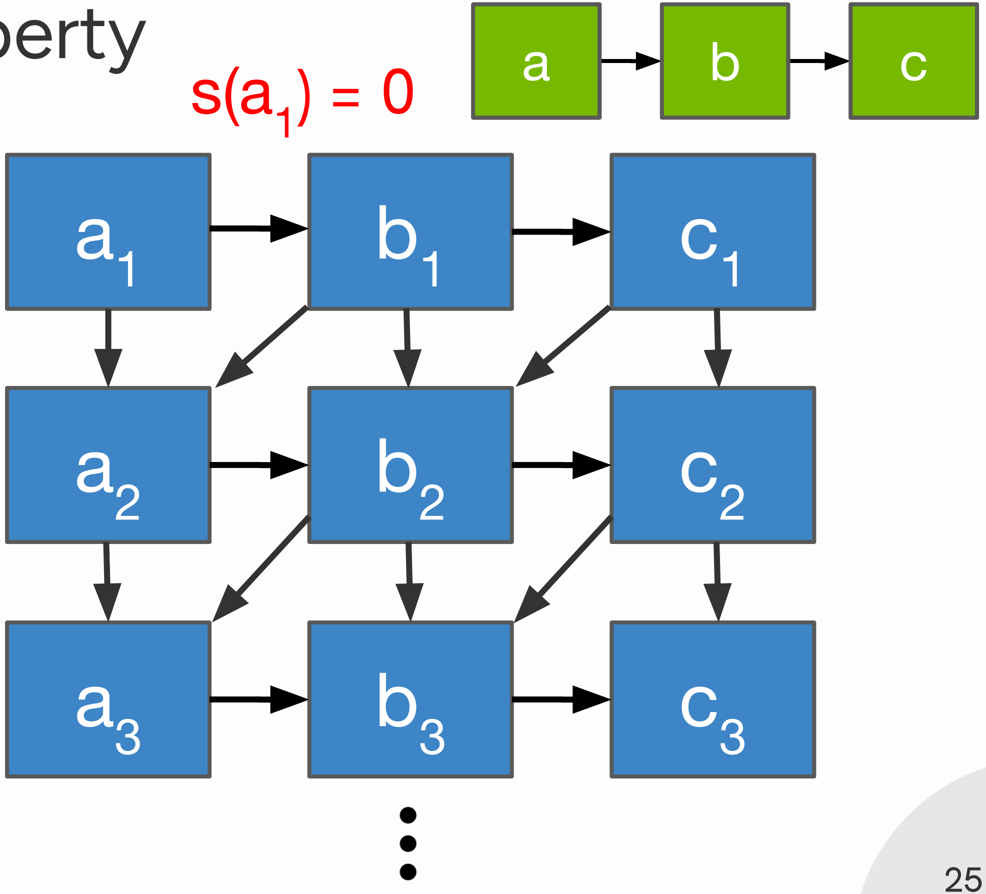
Key Trace Graph Property



- Longest path from source (a_1) defines a node's start time
  - Preconditions must be met before node may execute

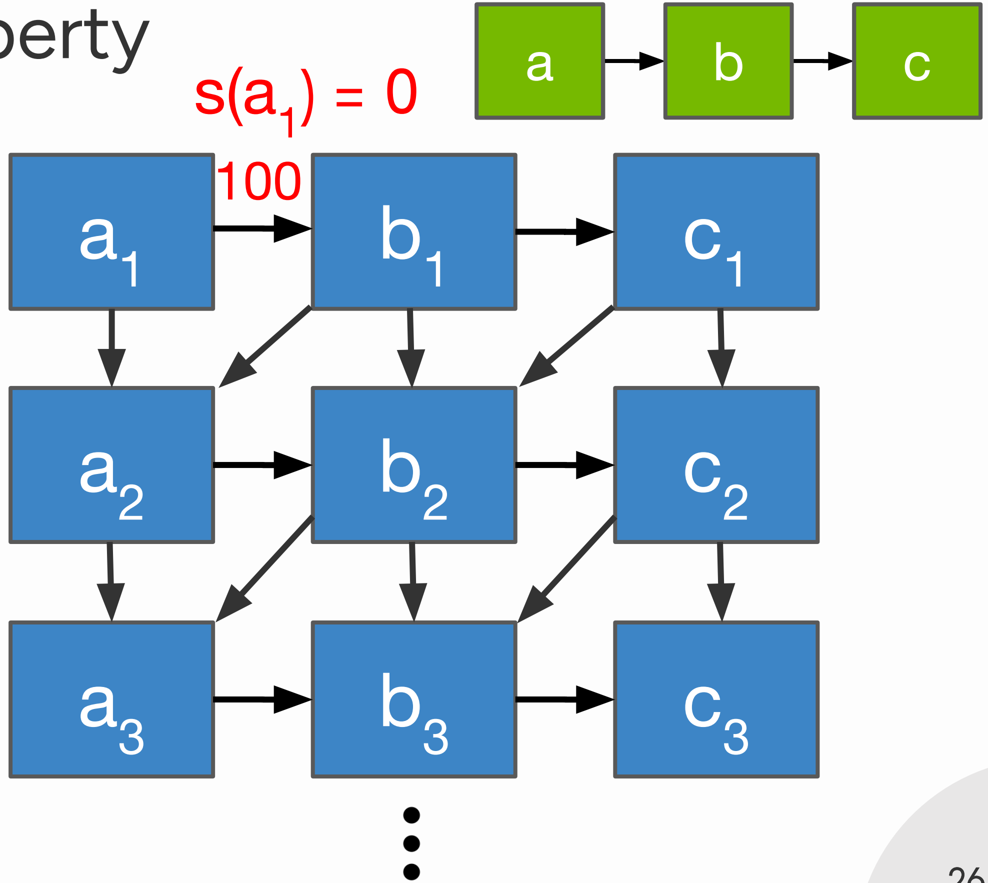
Key Trace Graph Property

- Longest path from source (a_1) defines a node's start time
  - Preconditions must be met before node may execute



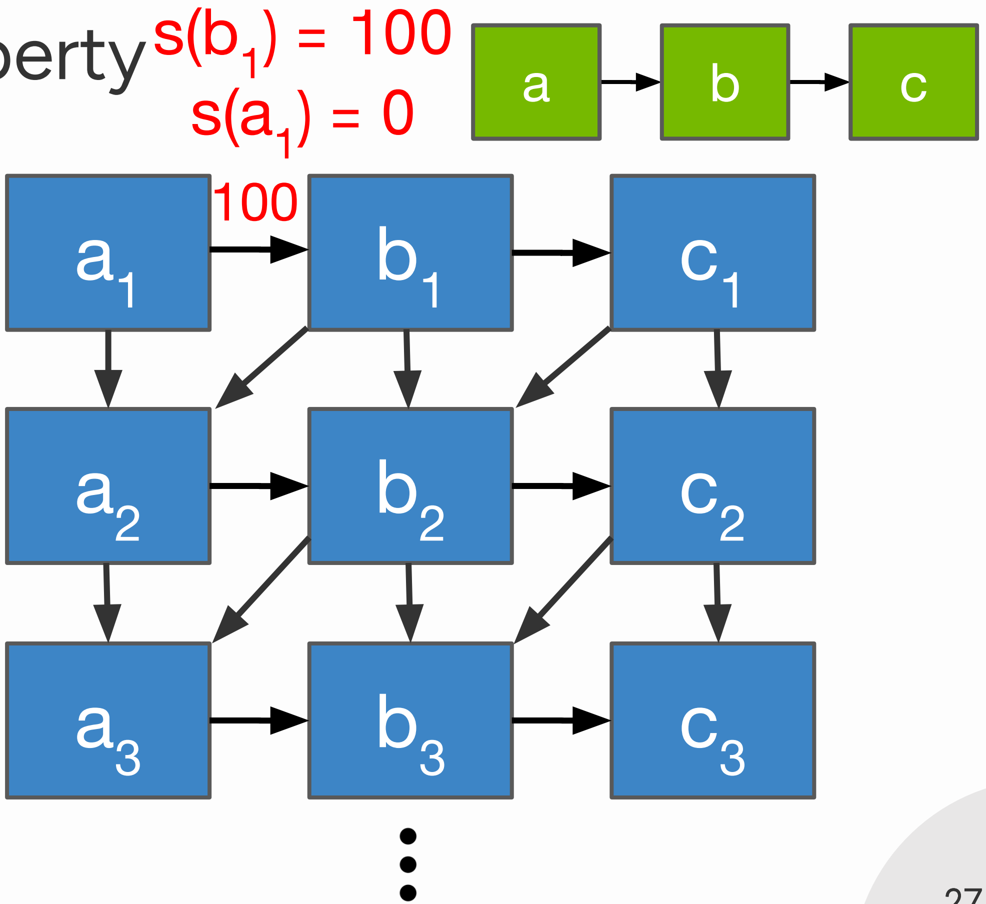
Key Trace Graph Property

- Longest path from source (a_1) defines a node's start time
  - Preconditions must be met before node may execute



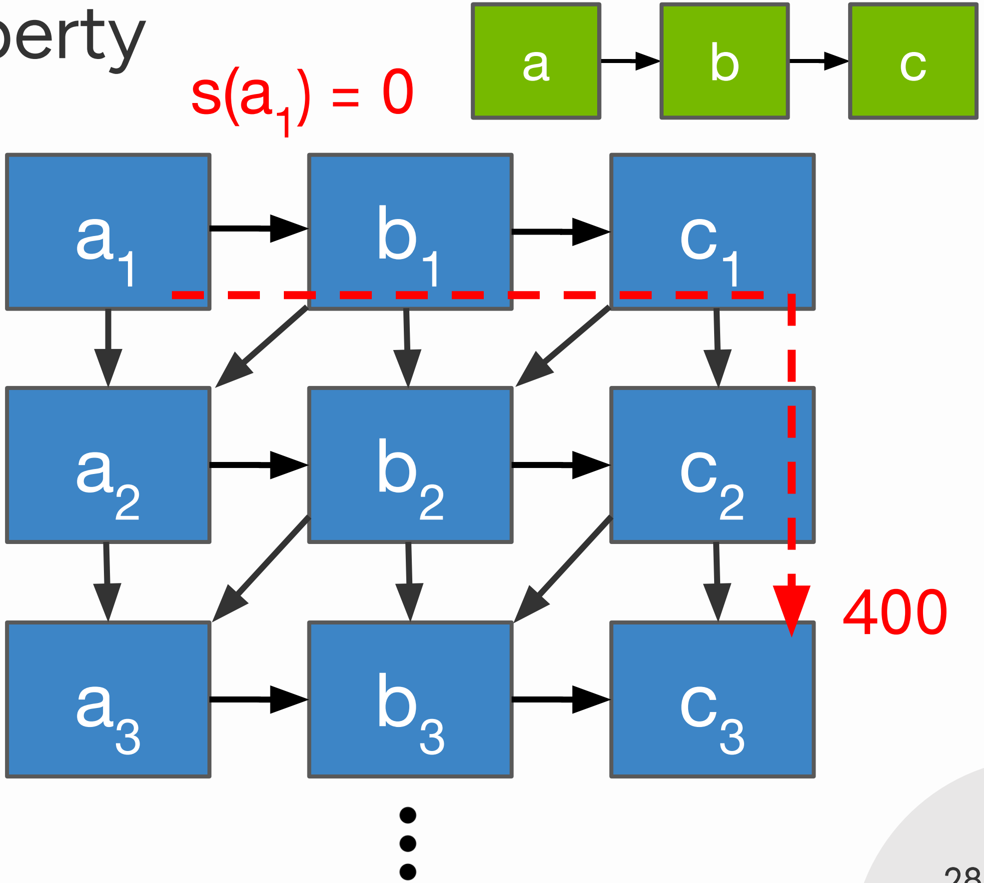
Key Trace Graph Property

- Longest path from source (a_1) defines a node's start time
  - Preconditions must be met before node may execute



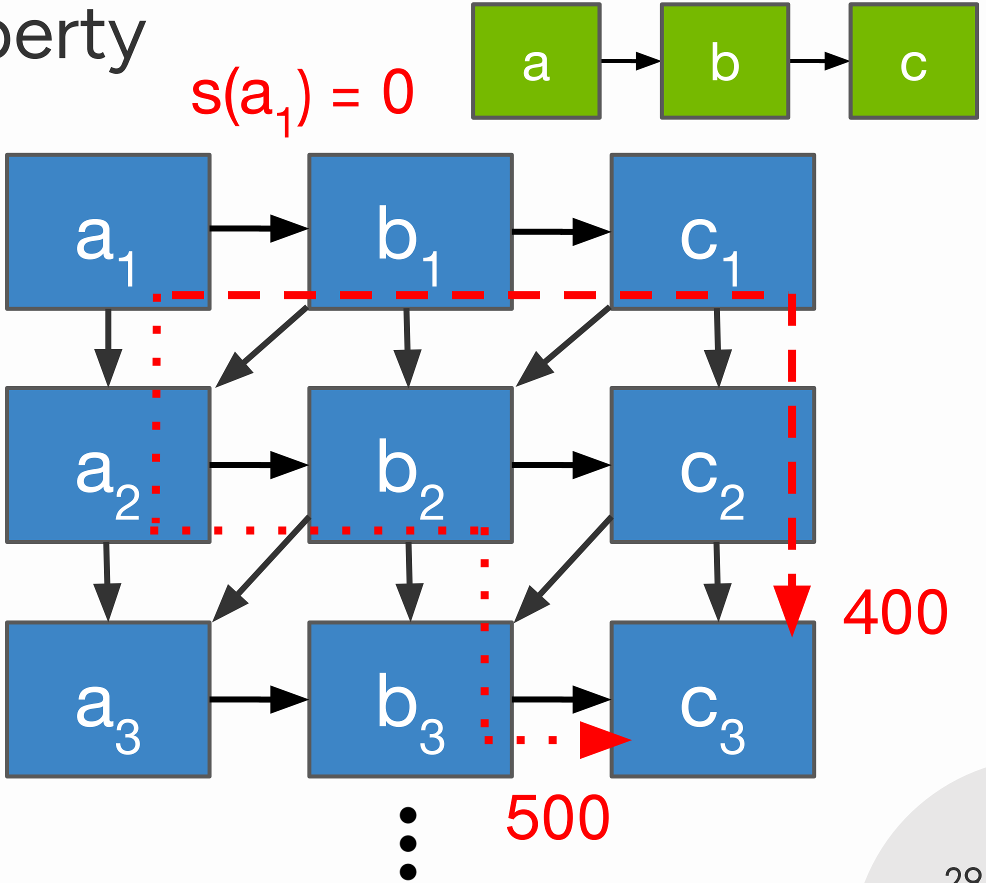
Key Trace Graph Property

- Longest path from source (a_1) defines a node's start time
  - Preconditions must be met before node may execute



Key Trace Graph Property

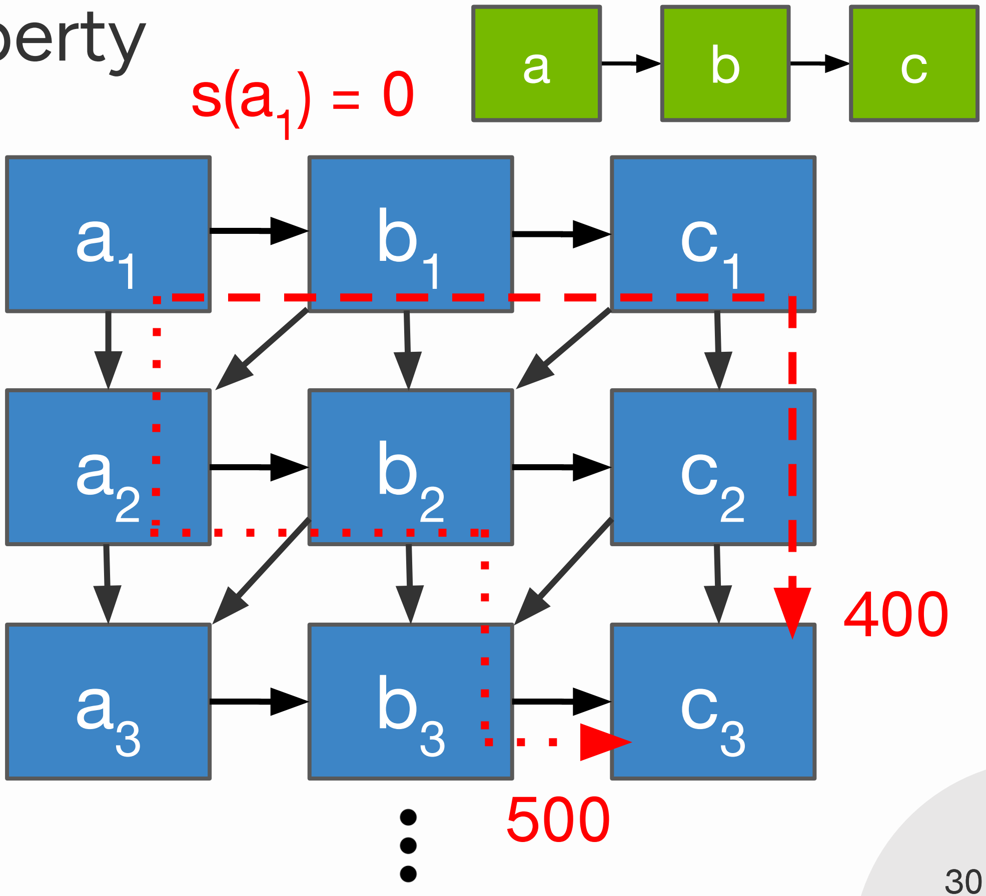
- Longest path from source (a_1) defines a node's start time
  - Preconditions must be met before node may execute



Key Trace Graph Property

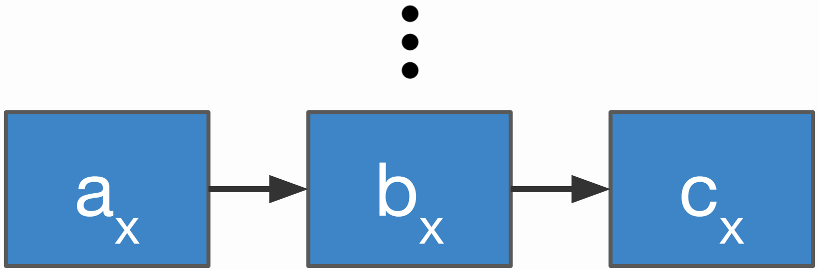
- Longest path from source (a_1) defines a node's start time
  - Preconditions must be met before node may execute

$s(c_3) = 500$



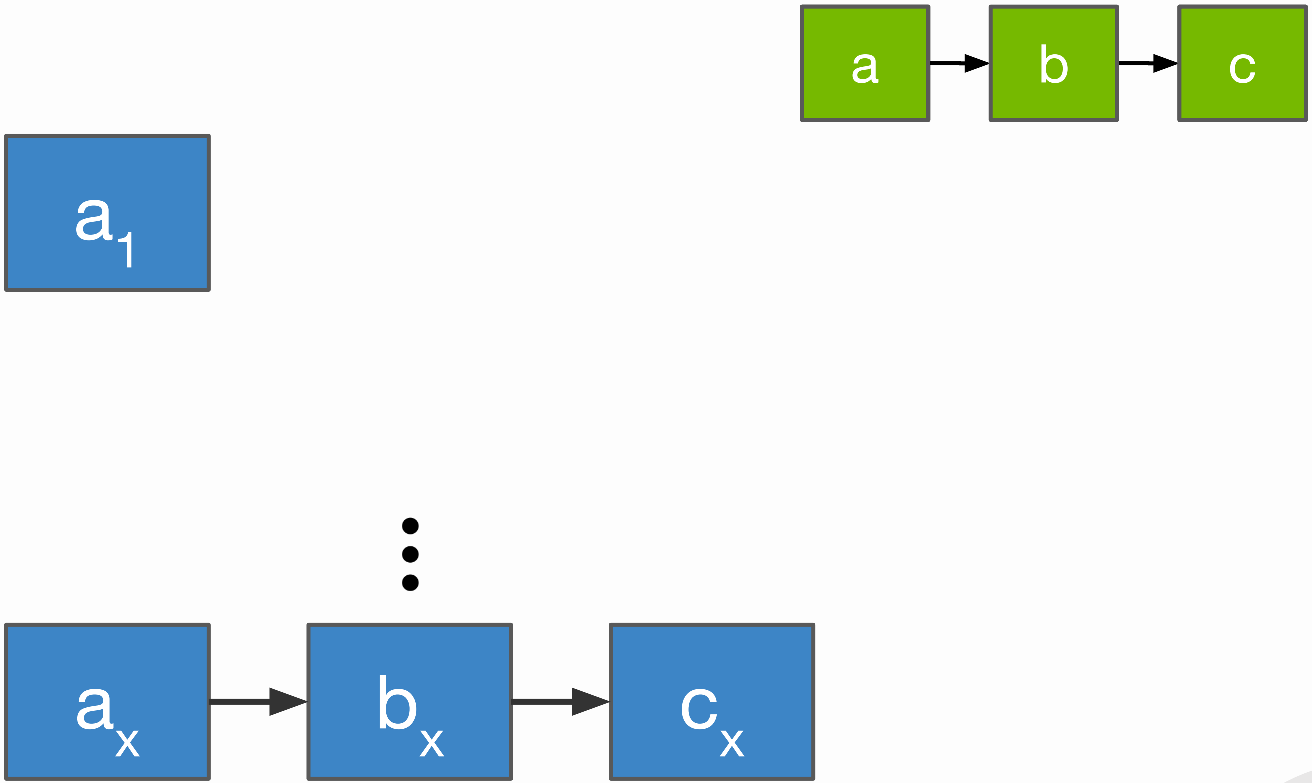
How can we leverage the trace graph for a response-time bound?

Key Idea



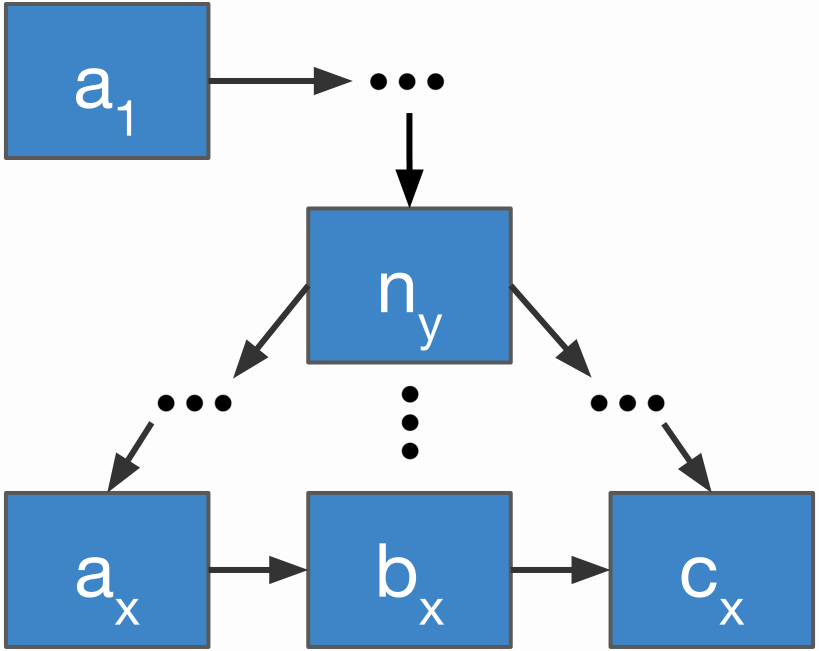
Assume the WCRT happens in iteration x

Key Idea



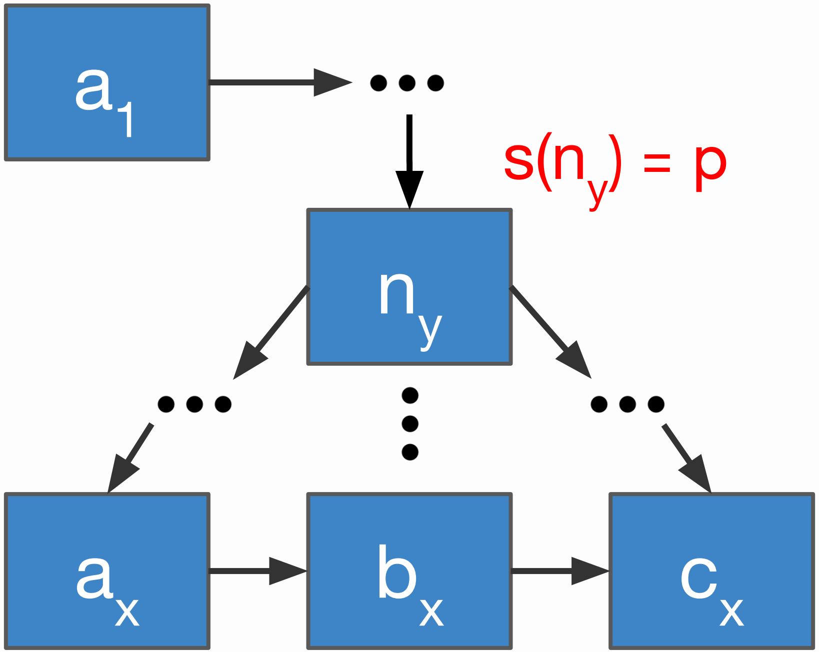
We know a_1 is somewhere in the past

Key Idea



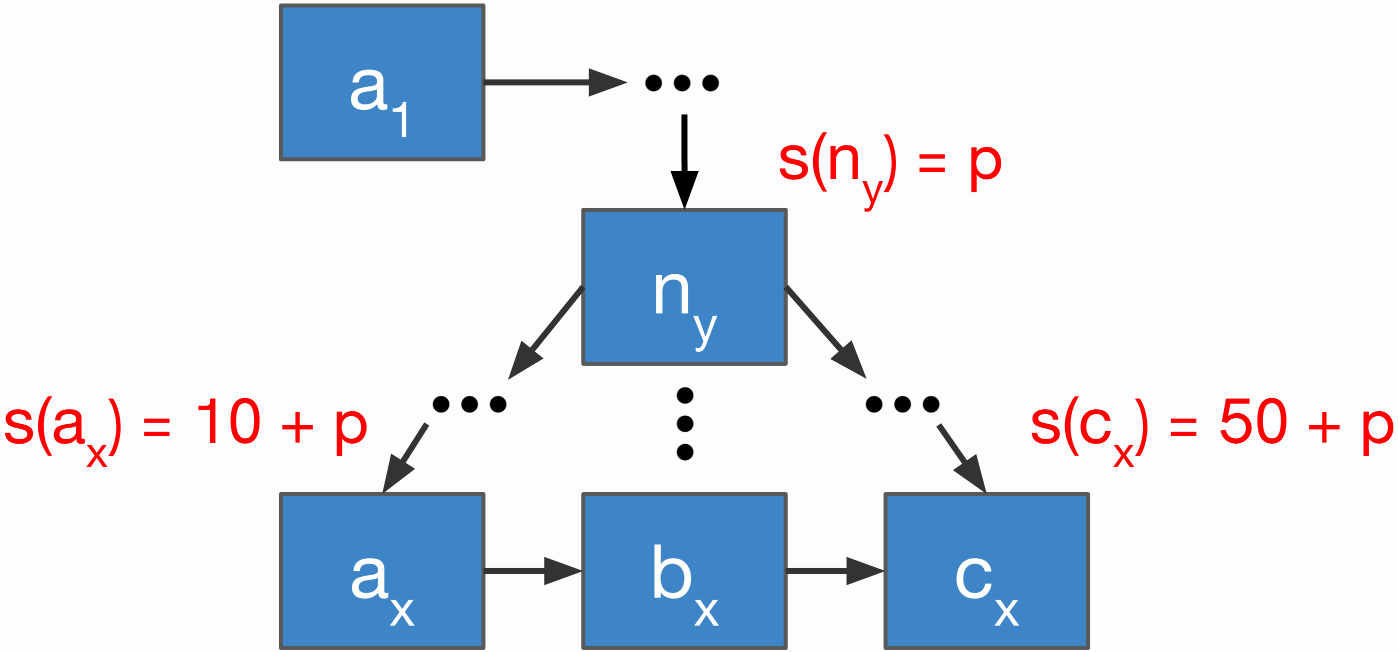
Find an n_y and assume on longest paths to a_x and c_x

Key Idea



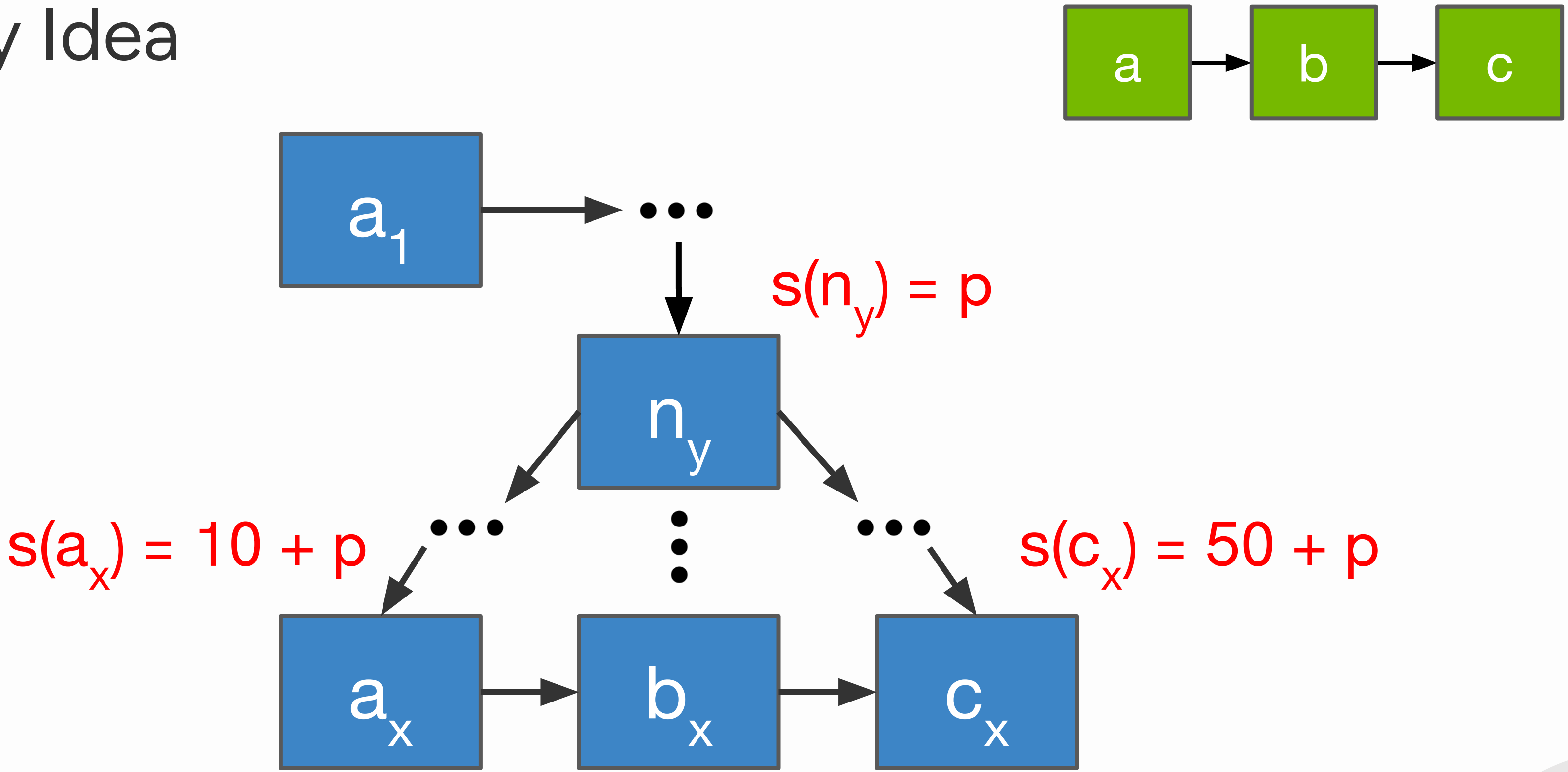
Longest path from a_1 to n_y can have any value

Key Idea



Compute path costs from n_y to a_x and c_x

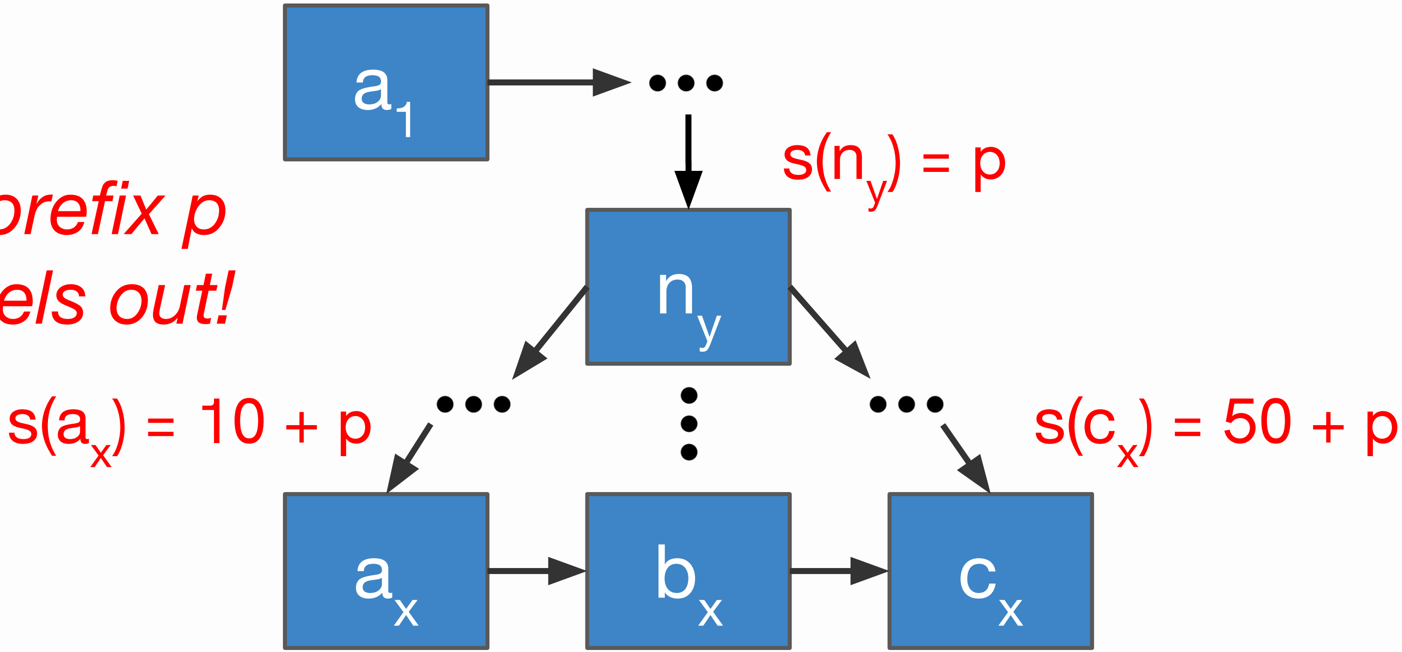
Key Idea



$s(c_x) - s(a_x) = (50 + p) - (10 + p) = 40$

Key Idea

The prefix p cancels out!



$$s(c_x) - s(a_x) = (50 + p) - (10 + p) = 40$$

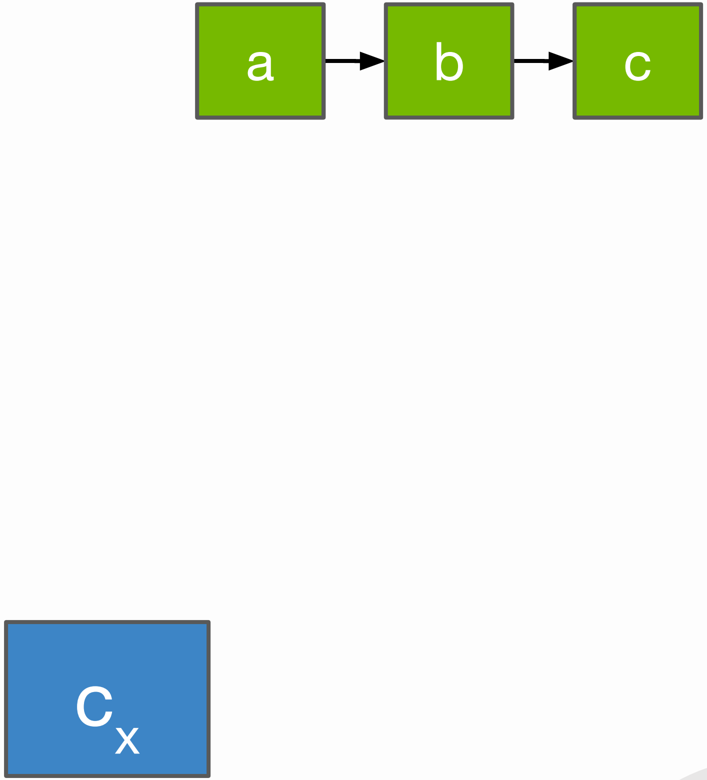
How can we leverage this insight to get a simple response-time algorithm?

- High-level overview: Start from an arbitrary iteration x, backtrack to find shared ancestors of the iteration x source and sink, and take differences of the paths from ancestor to source
  - Loop over the set of most recent shared ancestors

Algorithm

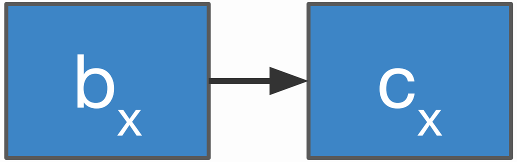
1. Find most recent shared ancestors
    a. Lemma: all paths to c_x from a_1 have an ancestor of a_x within a bounded number of iterations from x

Search For Ancestors



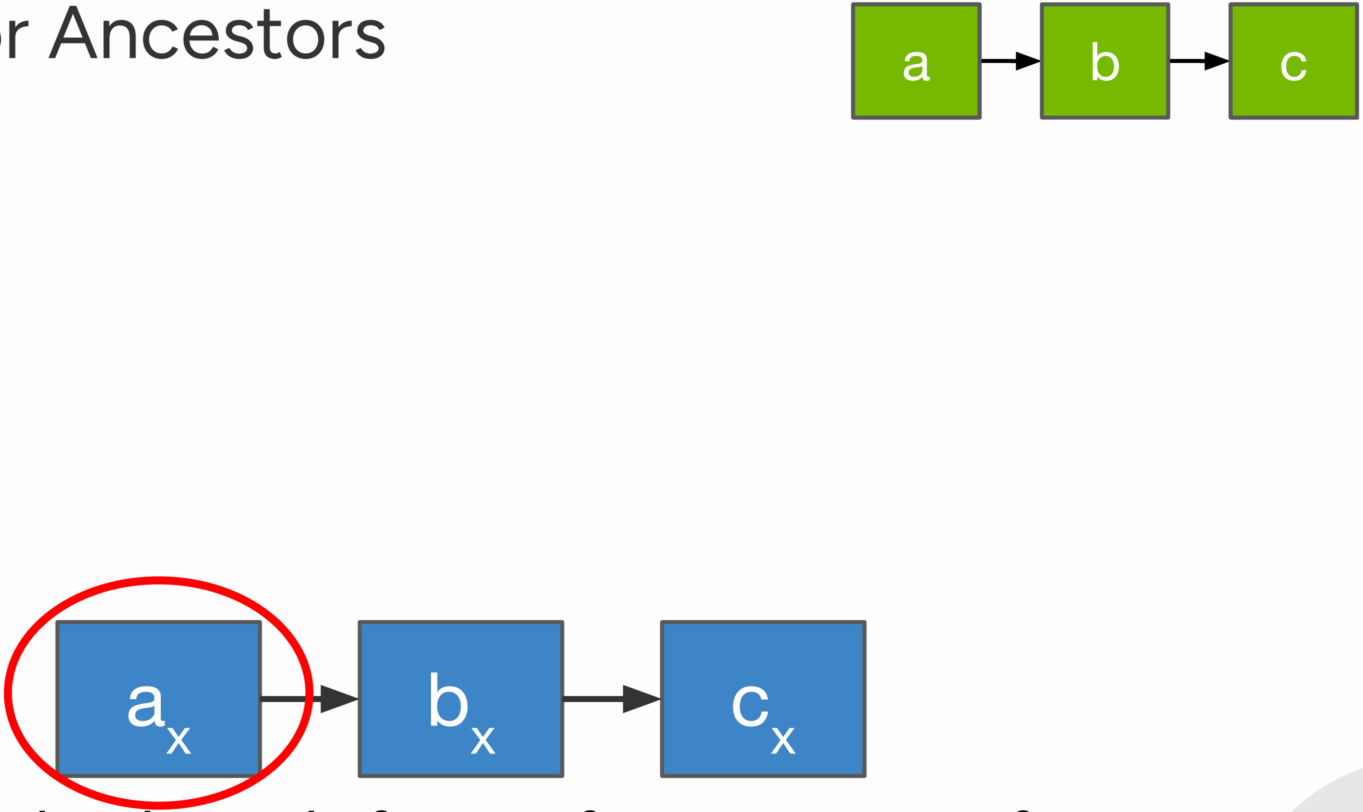
Search backwards from c_x for ancestors of a_x

Search For Ancestors



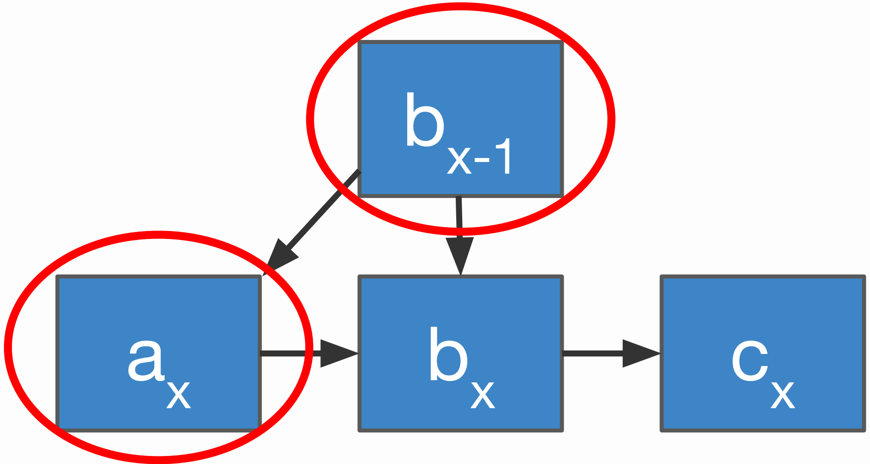
Search backwards from c_x for ancestors of a_x

Search For Ancestors



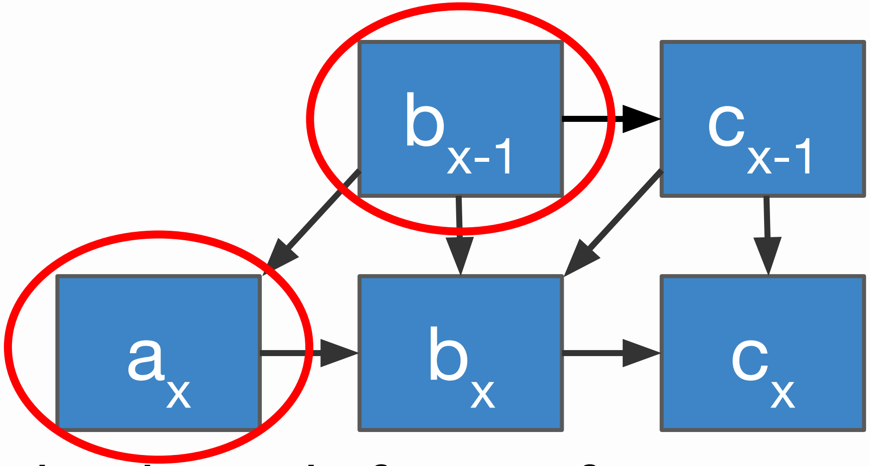
Search backwards from c_x for ancestors of a_x

Search For Ancestors



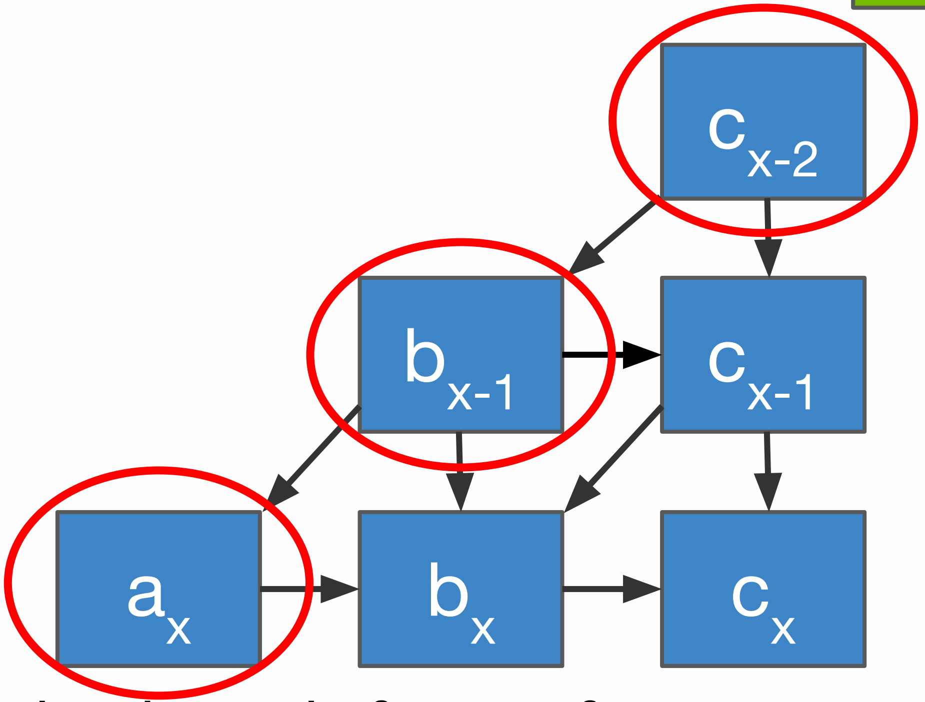
Search backwards from c_x for ancestors of a_x

Search For Ancestors



Search backwards from c_x for ancestors of a_x

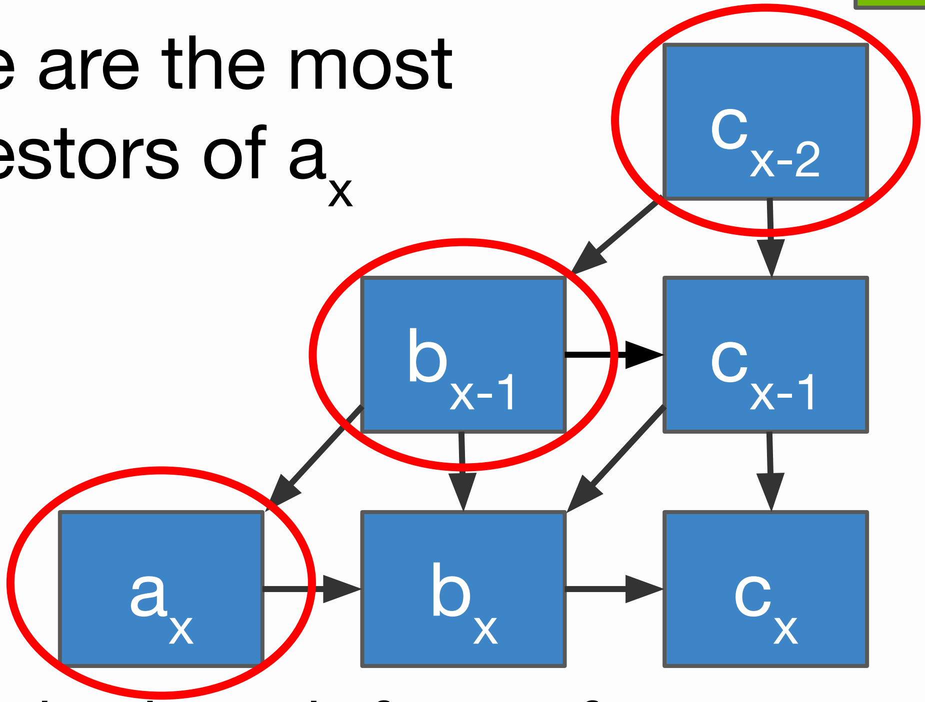
Search For Ancestors



Search backwards from c_x for ancestors of a_x

Search For Ancestors

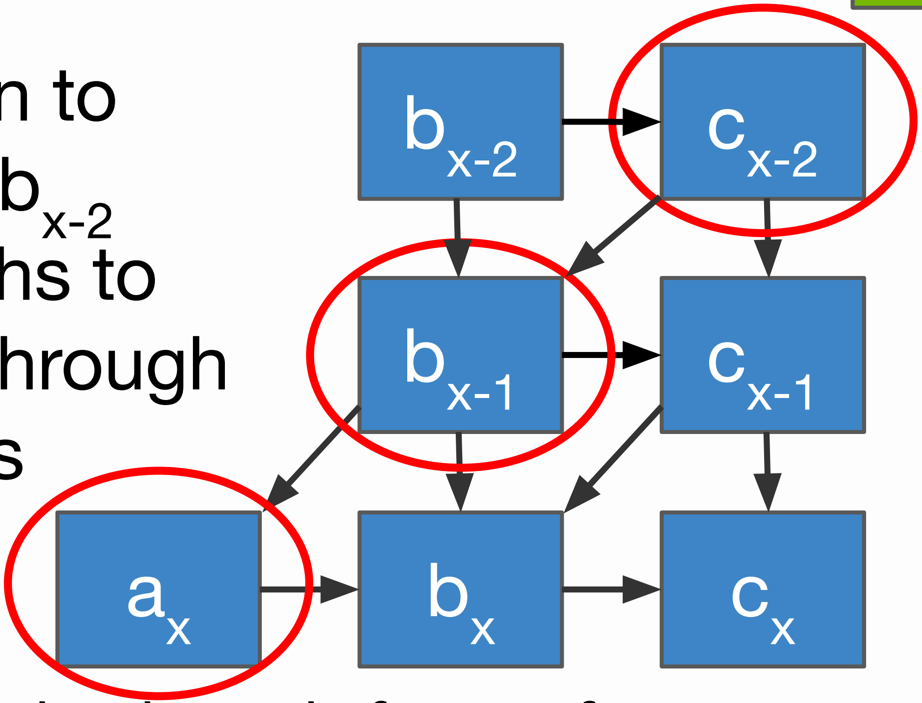
These three are the most recent ancestors of a_x



Search backwards from c_x for ancestors of a_x

Search For Ancestors

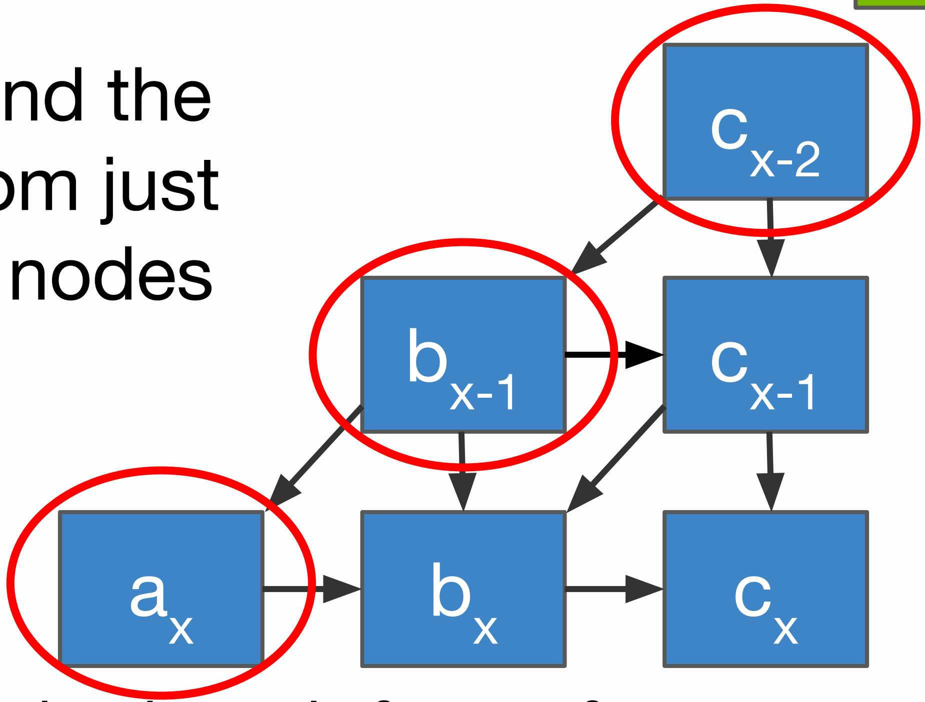
No reason to consider b_{x-2} as its paths to a_x/c_x go through the others



Search backwards from c_x for ancestors of a_x

Search For Ancestors

We can find the WCRT from just these six nodes

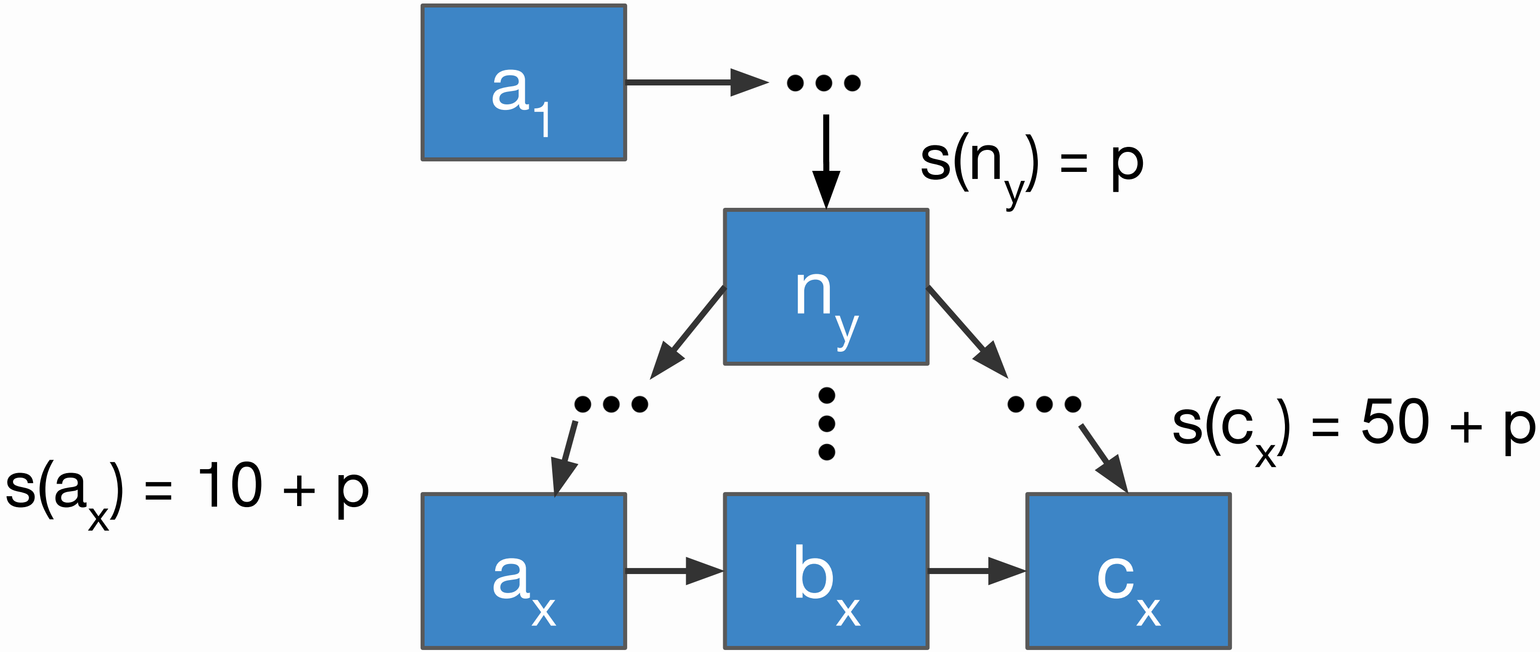


Search backwards from c_x for ancestors of a_x

Algorithm

1. Find most recent shared ancestors
    a. Lemma: all paths to c_x from a_1 have an ancestor of a_x within a bounded number of iterations from x
2. **Get the response time assuming each ancestor found in step 1 is on the longest path to c_x**

Response-Time Bound

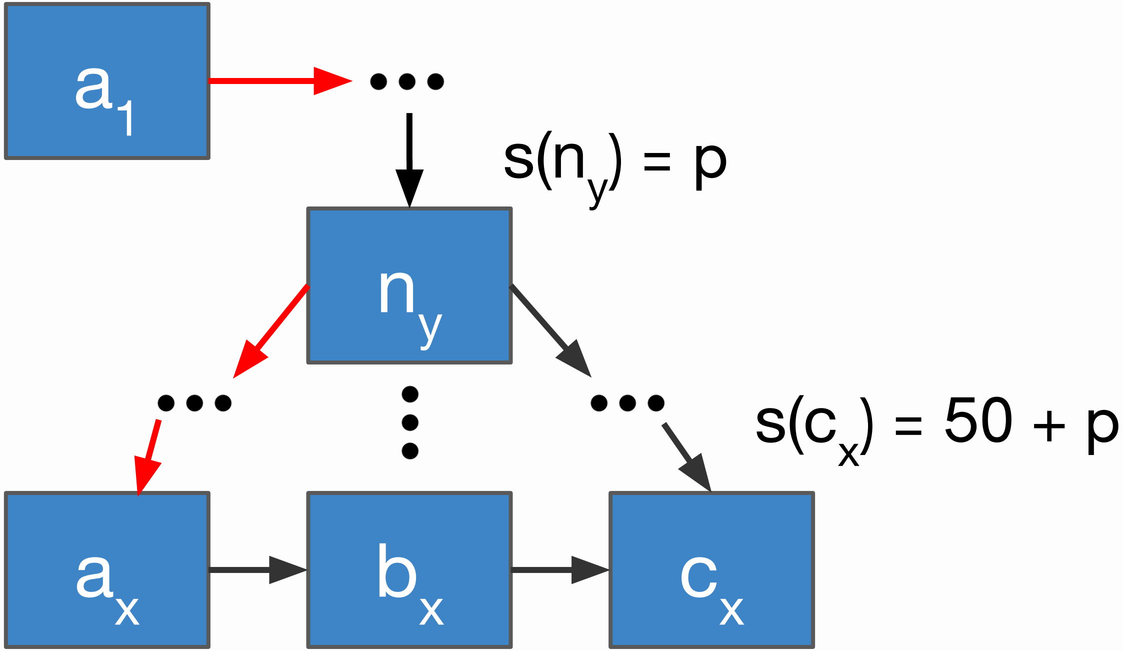


$$s(c_x) - s(a_x) = (50 + p) - (10 + p) = 40$$

Response-Time Bound

Assumed longest path to a_x came from n_y!

$s(a_x) = 10 + p$

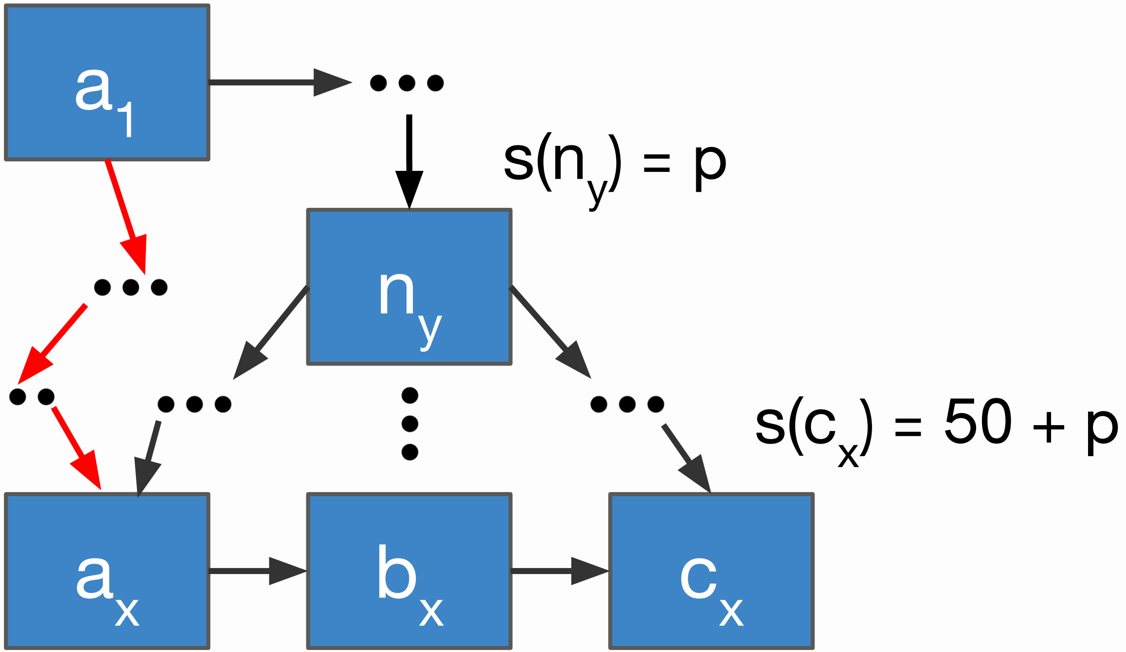


$s(c_x) - s(a_x) = (50 + p) - (10 + p) = 40$

Response-Time Bound

Assumed longest path to a_x came from n_y!

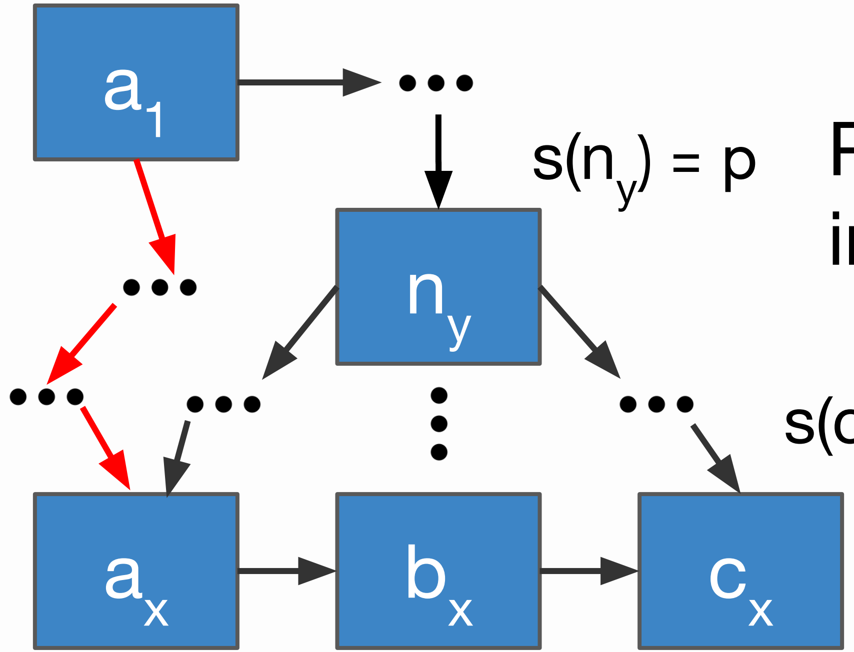
$s(a_x) = 10 + p$



$s(c_x) - s(a_x) = (50 + p) - (10 + p) = 40$

Response-Time Bound

Assumed longest path to a_x came from n_y!



$s(n_y) = p$

$s(a_x) = 10 + p$

$s(c_x) = 50 + p$

Proof of safety in paper

$s(c_x) - s(a_x) = (50 + p) - (10 + p) = 40$

Algorithm

1. Find most recent shared ancestors
    a. Lemma: all paths to c_x from a_1 have an ancestor of a_x within a bounded number of iterations from x
2. Get the response time assuming each ancestor found in step 1 is on the longest path to c_x
3. **Max of all candidate response times is WCRT**

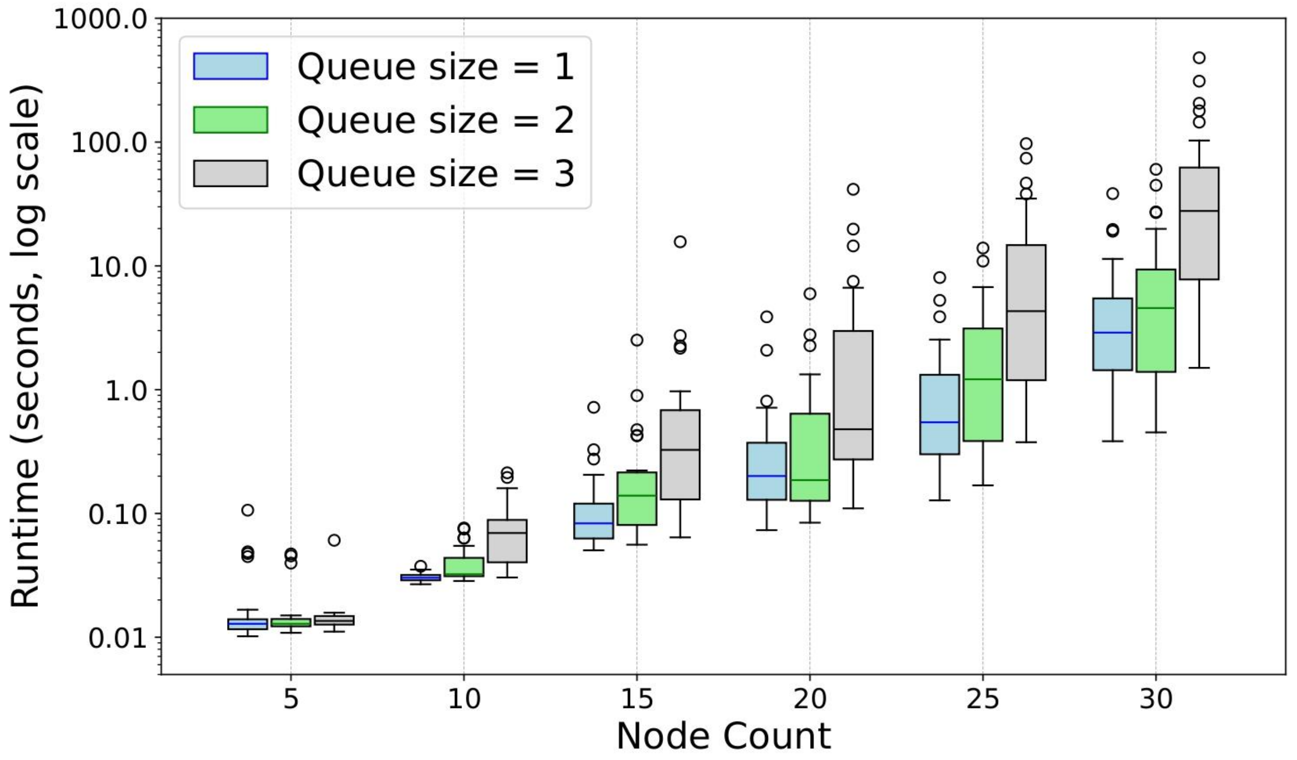
Exactness

- Our response-time bound is also exact (proof in paper)
  - We create a configuration of valid edge costs that yield the same response-time as our bound (needs 0 execution times)
- Supports variable execution time
- Supports variable queue sizes
  - This and other modifications are possible by altering the rules of how edges are drawn between nodes

Scalability

UPPAAL: 6000 seconds
for 9 node graph

Unoptimized
algorithm



https://github.com/ubc-systopia/rt-holoscan-artifacts

Conclusion

- Novel response-time analysis for DAGs with static backpressure: exact, fast, variable execution time
- Show equivalence between DAG model and SDFGs
  - Can solve problem via model checking, but this is slow
- WiP: Incorporating GPU interactions
- Future work: Scheduling

https://github.com/ubc-systopia/rt-holoscan-artifacts

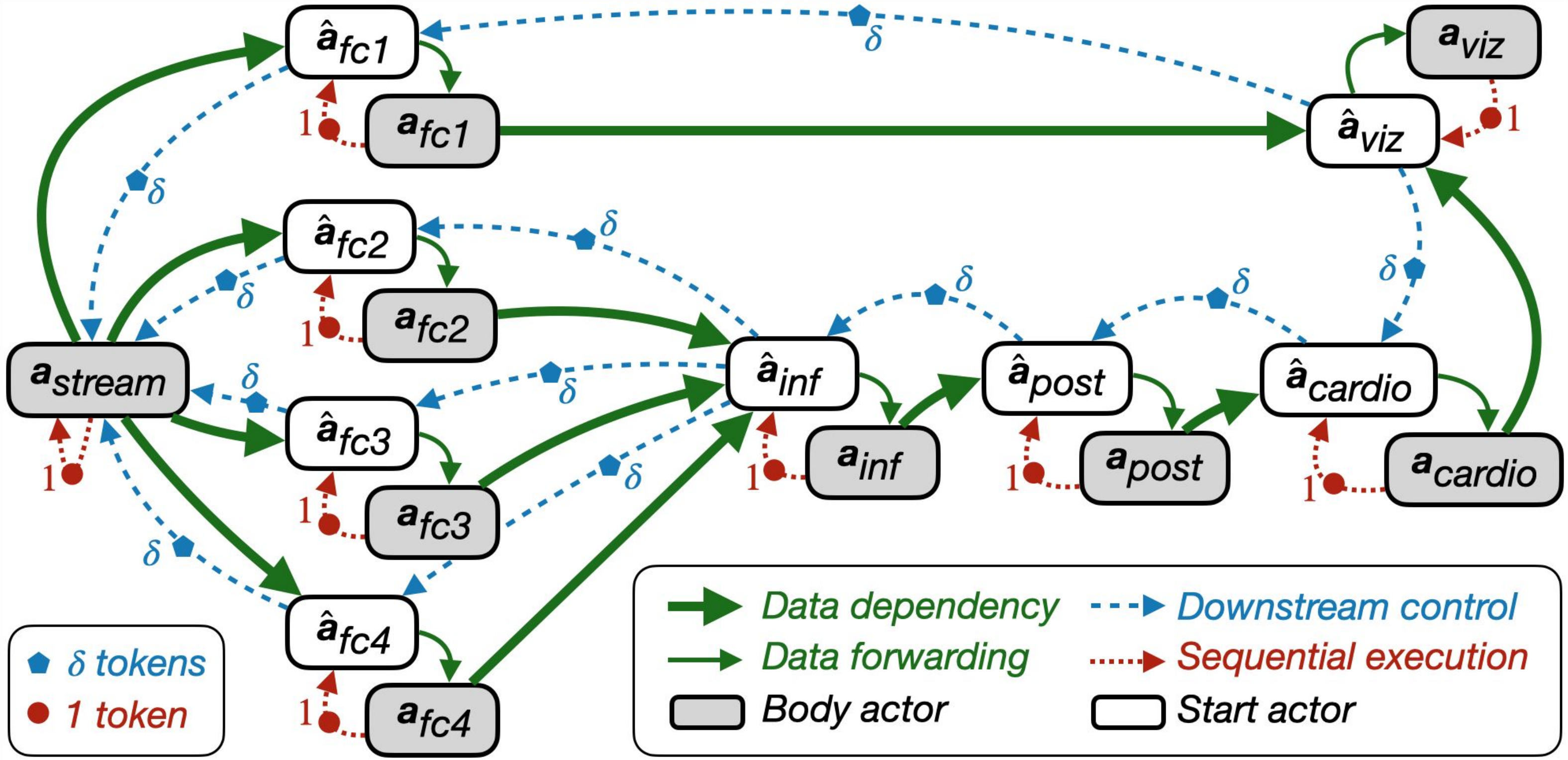
Conclusion

Thank you for listening! Questions?

- Novel response-time analysis for DAGs with static backpressure: exact, fast, variable execution time
- Show equivalence between DAG model and SDFGs
  - Can solve problem via model checking, but this is slow
- WiP: Incorporating GPU interactions
- Future work: Scheduling

Backup

SDFG

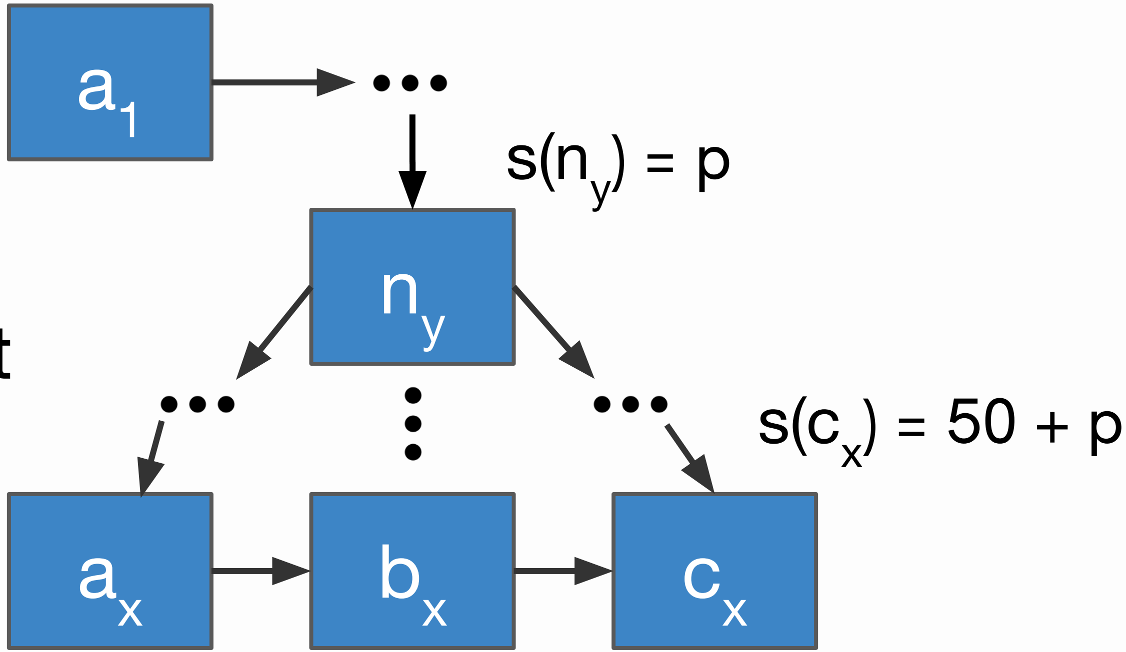


$\hat{a}_{fc1}$
a_{fc1}
a_{viz}
$\hat{a}_{viz}$
$\hat{a}_{fc2}$
a_{fc2}
a_{stream}
$\hat{a}_{fc3}$
a_{fc3}
$\hat{a}_{inf}$
a_{inf}
$\hat{a}_{post}$
a_{post}
$\hat{a}_{cardio}$
a_{cardio}
$\hat{a}_{fc4}$
a_{fc4}
δ
1
δ tokens
1 token
Data dependency
Downstream control
Data forwarding
Sequential execution
Body actor
Start actor

Response-Time Bound

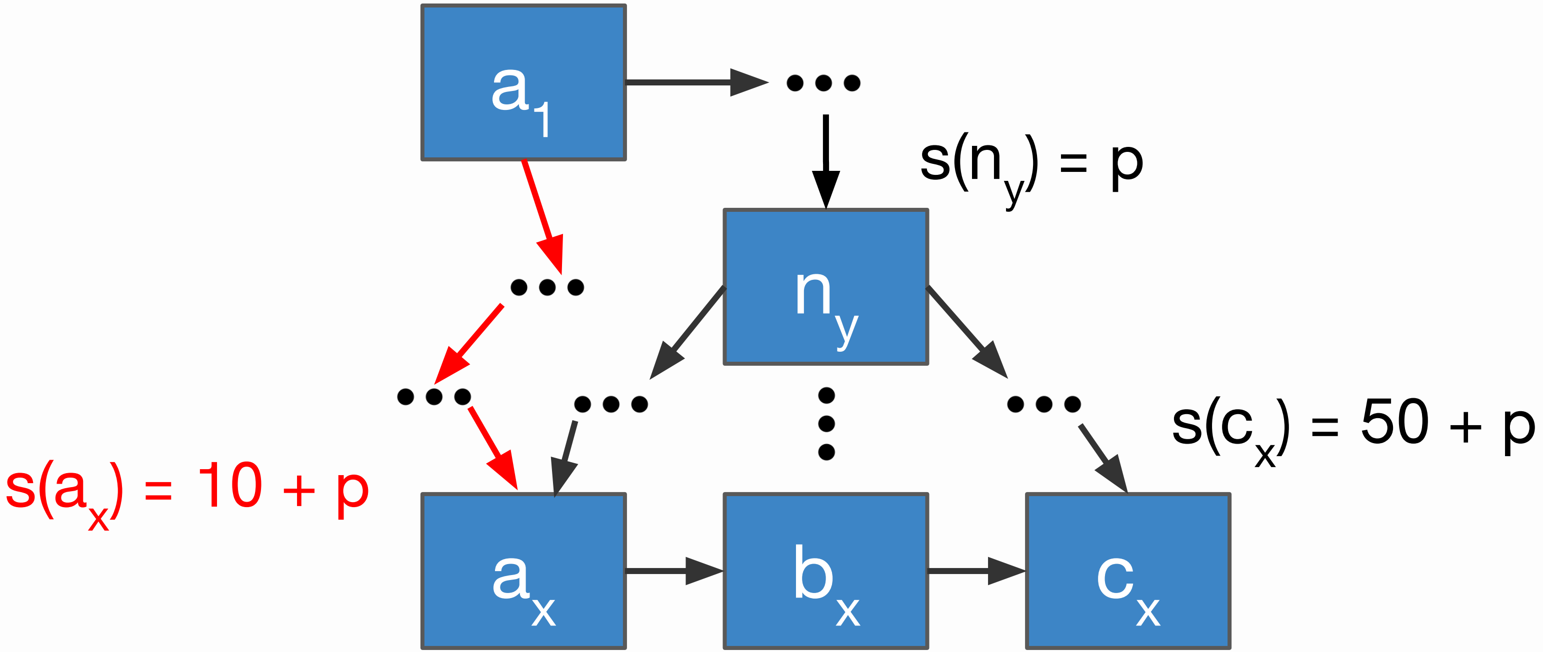
Two cases: longest path to a_x is either from n_y or not

$s(a_x) = 10 + p$



$s(c_x) - s(a_x) = (50 + p) - (10 + p) = 40$

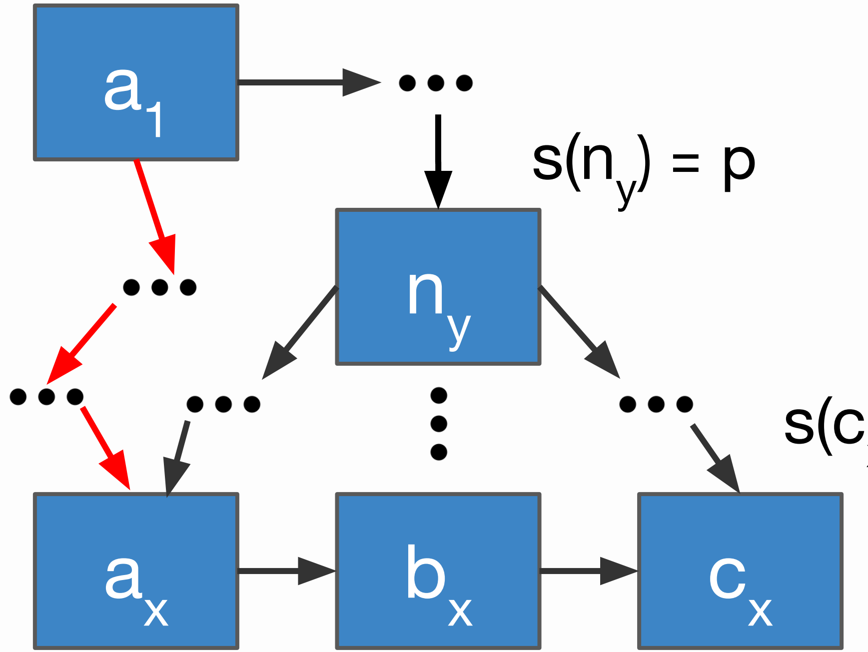
Response-Time Bound



$$s(c_x) - s(a_x) = (50 + p) - (10 + p) = 40$$

Response-Time Bound

New path with length q dominates



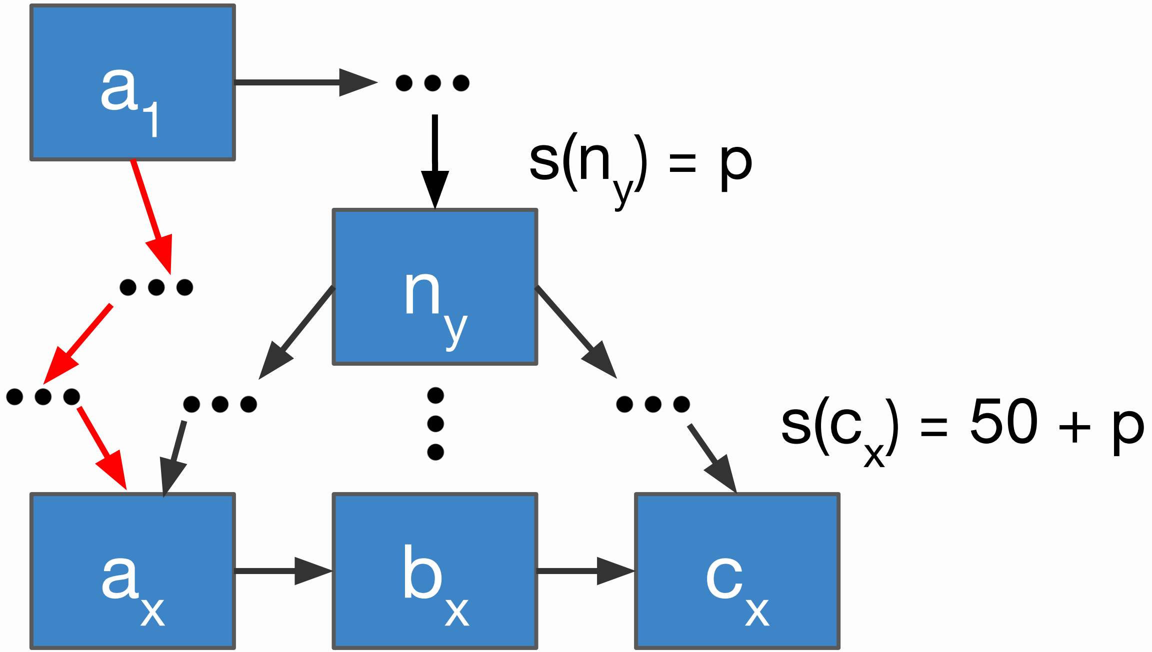
~~$s(a_x) = 10 + p$~~

$s(a_x) =$
$q > 10 + p$

$s(c_x) - s(a_x) = (50 + p) - (10 + p) = 40$

Response-Time Bound

But this *decreases* response time



~~$s(a_x) = 10 + p$~~

$s(a_x) =$
$q > 10 + p$

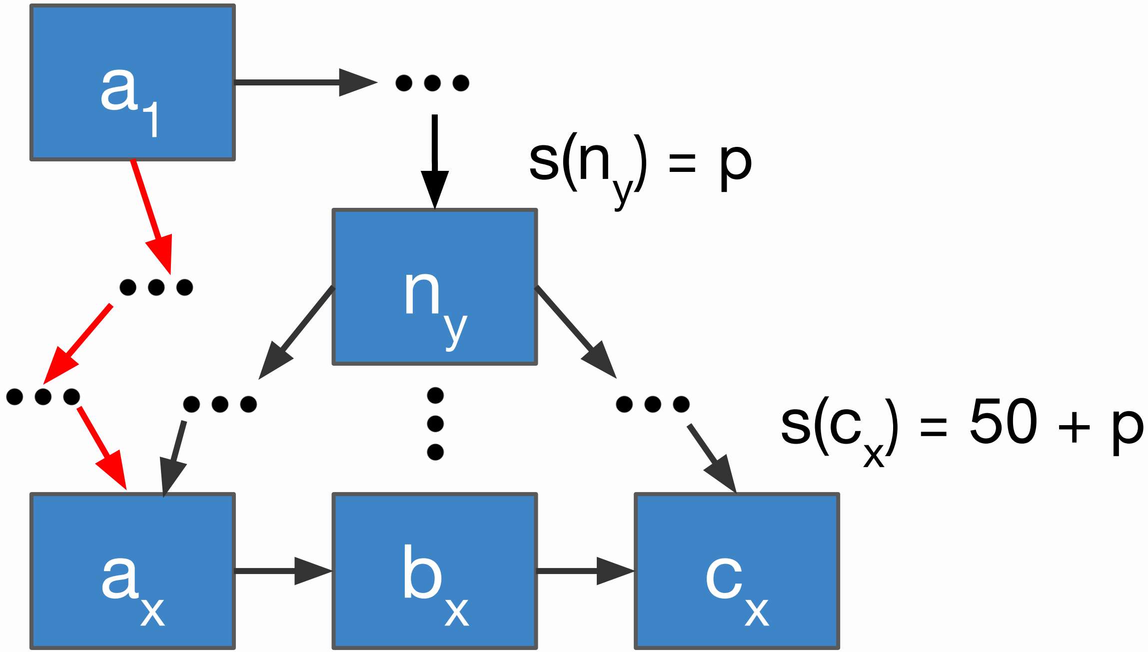
$s(c_x) - s(a_x) < (50 + p) - (10 + p) = 40$

Response-Time Bound

If n_y is on longest path to c_x:
RT <= 40

~~$s(a_x) = 10 + p$~~

$s(a_x) =$
$q > 10 + p$



$s(c_x) - s(a_x) < (50 + p) - (10 + p) = 40$