Worst-case memory latency and distributed DRAM banks across chips

Towards tighter bounds using Deterministic Network Calculus

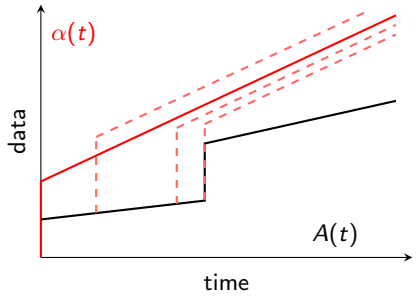
Edoardo Loni[1] Raffaele Zippo[1] Giovanni Stea[1] Matteo Andreozzi[2]

[1]University of Pisa

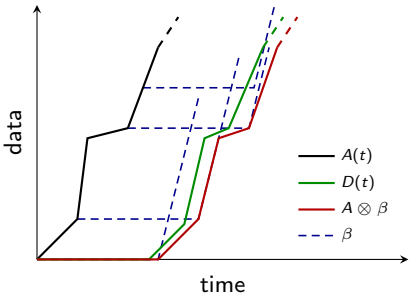
[2]ARM

Lund, RTSOPS 2026, 07-07-2026

Context: Deterministic Network Calculus



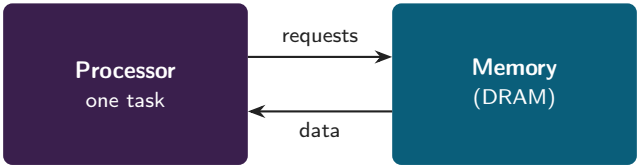
Arrival curve $\alpha(t)$: max traffic constraint;
$A(t) - A(s) \leq \alpha(t - s)$.



Service curve $\beta(t)$: min service guaranteed;
$D \geq A \otimes \beta$.

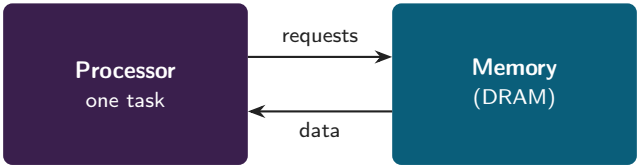
From α and β we derive worst-case latency (WCD) and worst-case backlog (WCB) bounds.

A simple model to start with



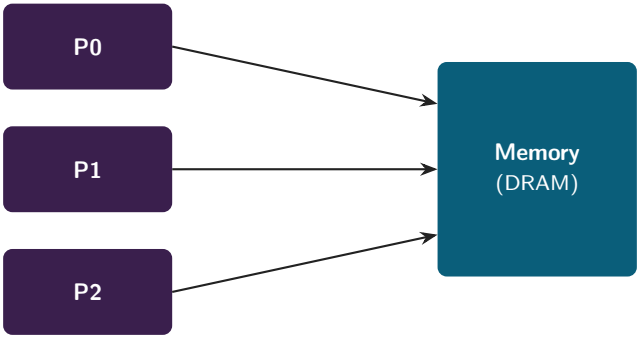
A simple computing system: one processor, running one task, that accesses the memory.

A simple model to start with



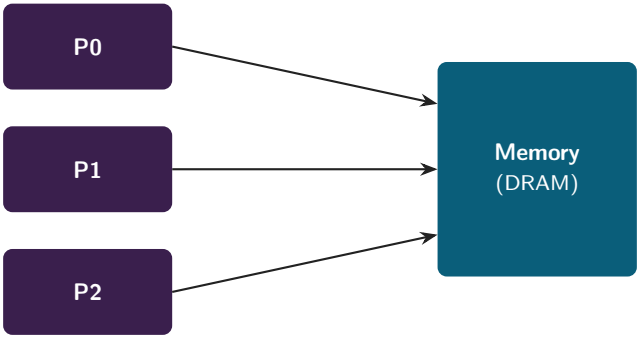
A simple computing system: one processor, running one task, that accesses the memory.
Only need to derive α for the requests and β for the memory.

More processors, one memory



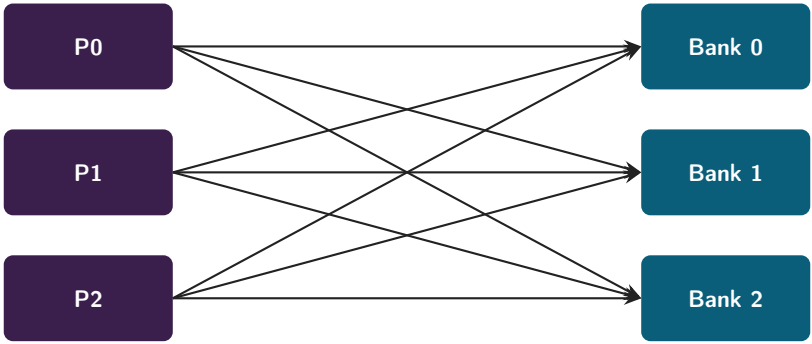
Concurrent processes compete for the same service.
Can be addressed using *residual service curves*, $\beta_2 = [\beta - (\alpha_0 + \alpha_1)]^+_\uparrow$.

More processors, one memory



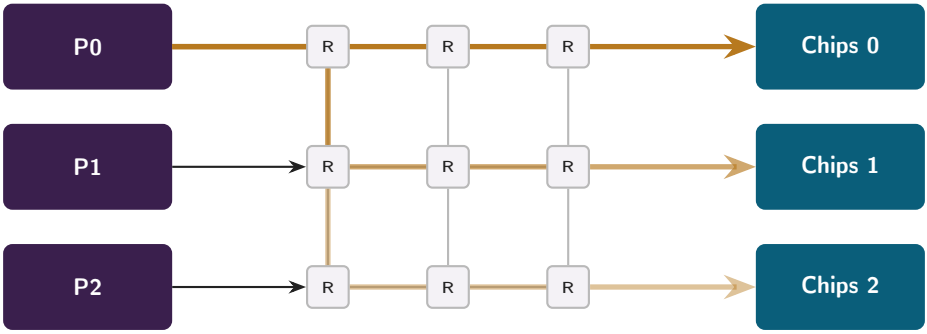
Concurrent processes compete for the same service.
Can be addressed using *residual service curves*, $\beta_2 = [\beta - (\alpha_0 + \alpha_1)]^+_\uparrow$.
An obvious bottleneck: DRAM is multiple chips, banks etc. accessed in parallel.

Requests spread across DRAM banks . . .



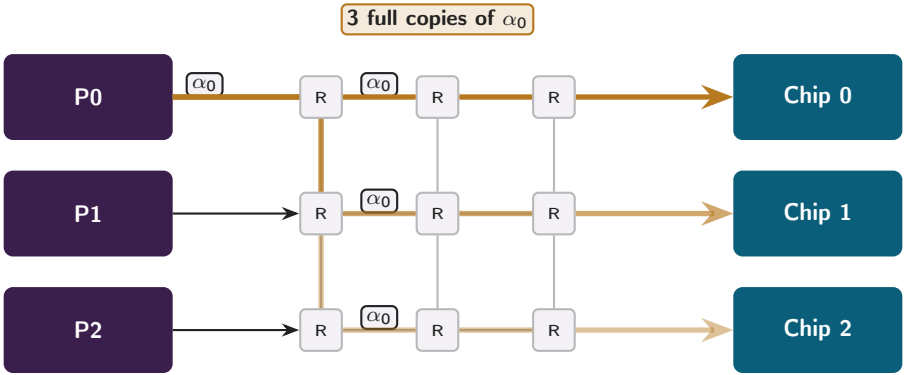
Each processor's traffic spreads to different banks.
This leads to more parallelism and less contention on average.

. . . located on different chips on a Network-on-Chip



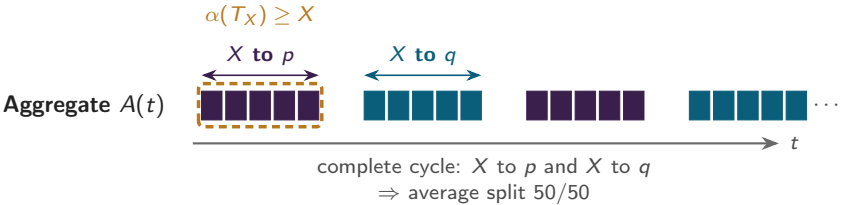
Traffic contention happens also on the NoC, and with other traffic going through it.
Tight traffic modelling is critical to bound the interconnect latencies.

State of the art: pessimistic splitting

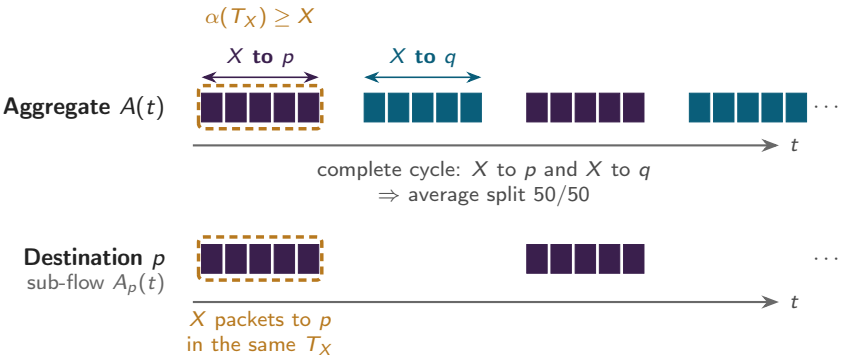


SOTA: replicated in full each processor's aggregate α^i to every bank; pessimism compounds at *every* split point.

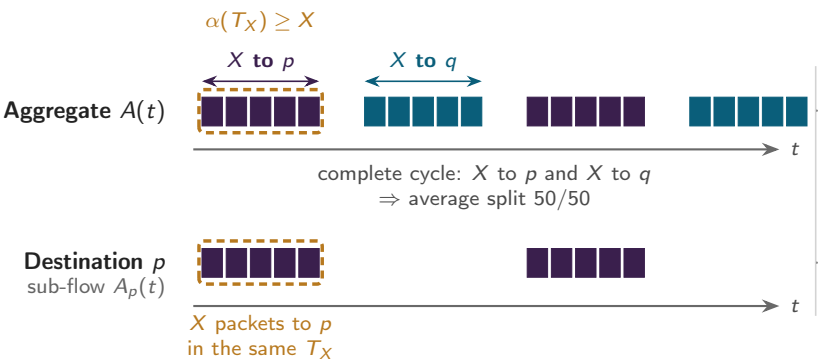
State of the art: pessimistic splitting



State of the art: pessimistic splitting



State of the art: pessimistic splitting



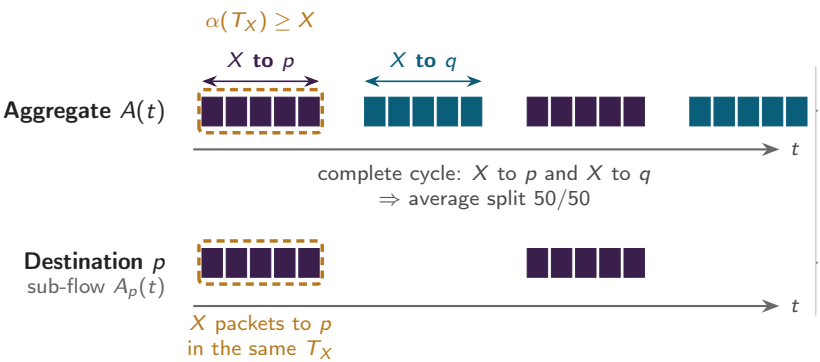
What a safe α_p must cover

In the highlighted window,
$A_p(t + T_X) - A_p(t) = X$.

Therefore
$\alpha_p(T_X) \geq X$.

✕ any $\tilde{\alpha}_p(T_X) < X$ is falsified

State of the art: pessimistic splitting



What a safe α_p must cover

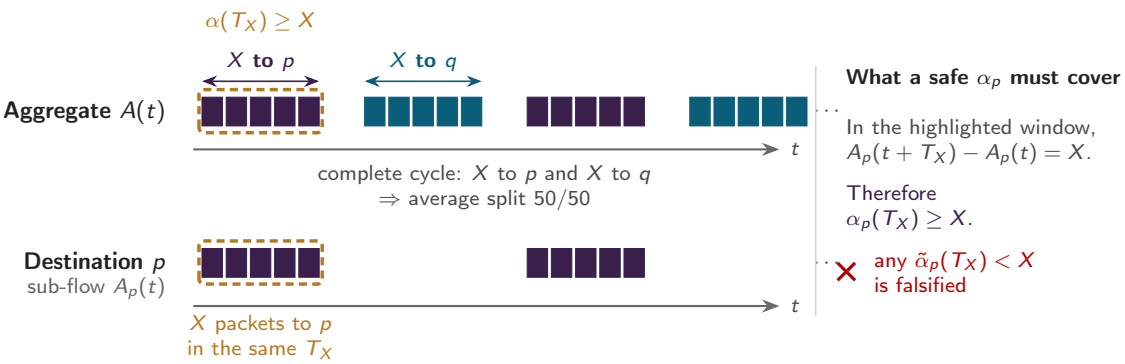
In the highlighted window,
$A_p(t + T_X) - A_p(t) = X$.

Therefore
$\alpha_p(T_X) \geq X$.

✗ any $\tilde{\alpha}_p(T_X) < X$ is falsified

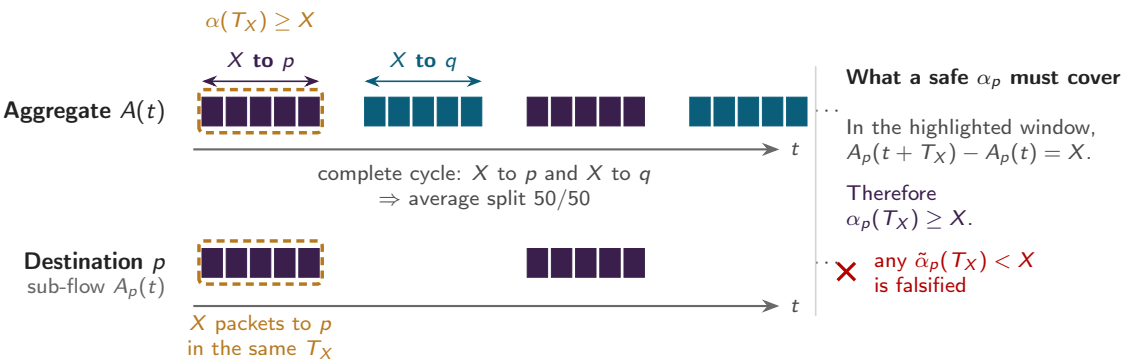
- Knowing the average ratio is 50/50 is not enough: $\forall X$, trace $p^X q^X p^X \ldots$ is admissible.

State of the art: pessimistic splitting



- Knowing the average ratio is 50/50 is not enough: $\forall X$, trace $p^X q^X p^X \ldots$ is admissible.
- Since X is arbitrary, then α_p must allow every burst allowed by α.

State of the art: pessimistic splitting



- Knowing the average ratio is 50/50 is not enough: $\forall X$, trace $p^X q^X p^X \ldots$ is admissible.
- Since X is arbitrary, then α_p must allow every burst allowed by α.
- Therefore the only safe α_p is α.

A bottom-up approach

The problem is practical, but the underlying issue is theoretical.
So we work on three levels

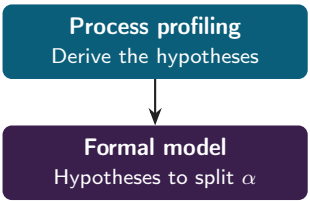
1. Which hypotheses does the formal model need to split an aggregate arrival curve?

Formal model
Hypotheses to split α

A bottom-up approach

The problem is practical, but the underlying issue is theoretical.
So we work on three levels

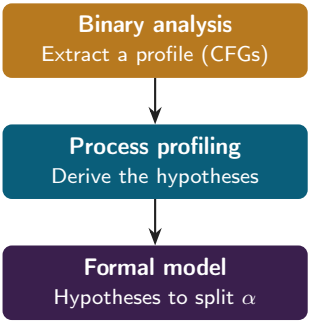
1. Which hypotheses does the formal model need to split an aggregate arrival curve?
2. Which process profiling is needed to derive these hypotheses?



A bottom-up approach

The problem is practical, but the underlying issue is theoretical.
So we work on three levels

1. Which hypotheses does the formal model need to split an aggregate arrival curve?
2. Which process profiling is needed to derive these hypotheses?
3. How do we analyze a given binary to extract such profile?



Formal model - key idea

- Counter-example is an adversarial access pattern:
  arbitrary-sized burst in the short term hiding the long-term benefits

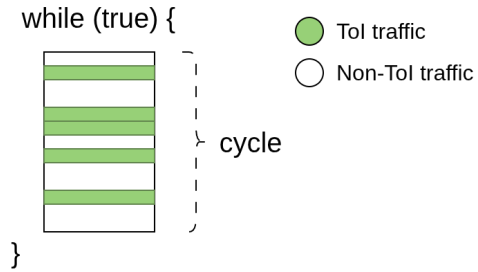
Formal model - key idea

- Counter-example is an adversarial access pattern:
  arbitrary-sized burst in the short term hiding the long-term benefits
- To improve WC analysis, we need to make that pattern unfeasible

Formal model - key idea

- Counter-example is an adversarial access pattern:
  arbitrary-sized burst in the short term hiding the long-term benefits
- To improve WC analysis, we need to make that pattern unfeasible
- We do so by adding constraints related to the program loop

The cycle model

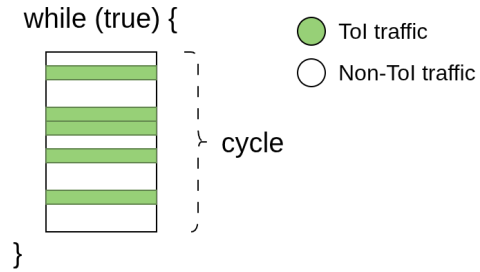


One cycle = traffic issued in one task iteration.

ToI stands for *Type of Interest*.

- Consider a (pseudo-)cyclic task: each iteration issues roughly the same *type* of traffic, i.e. predictable data structure accesses during each phase.

The cycle model

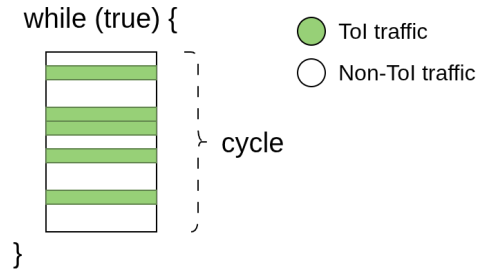


One cycle = traffic issued in one task iteration.

ToI stands for *Type of Interest*.

- Consider a (pseudo-)cyclic task: each iteration issues roughly the same *type* of traffic, i.e. predictable data structure accesses during each phase.
- Let $N^{\min}$ be the minimum *aggregate* requests in a cycle
- Let $N^{\max}_{\text{toi}}$ be the maximum amount of ToI requests in a cycle

The cycle model

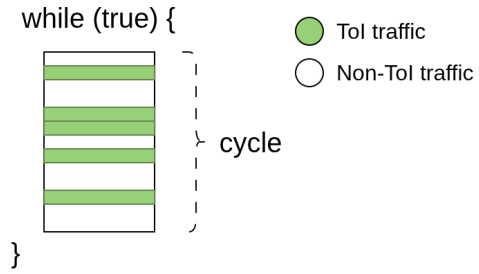


One cycle = traffic issued in one task iteration.

ToI stands for *Type of Interest*.

- Consider a (pseudo-)cyclic task: each iteration issues roughly the same *type* of traffic, i.e. predictable data structure accesses during each phase.
- Let $N^{\min}$ be the minimum *aggregate* requests in a cycle
- Let $N^{\max}_{\text{toi}}$ be the maximum amount of ToI requests in a cycle
- If $N^{\max}_{\text{toi}} < N^{\min}$, there is a minimum gap between each burst of ToI traffic, and arbitrarily long bursts are not possible

The cycle model



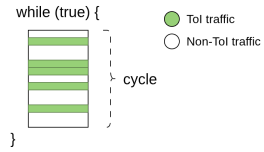
One cycle = traffic issued in one task iteration.

ToI stands for *Type of Interest*.

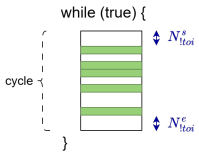
- Consider a (pseudo-)cyclic task: each iteration issues roughly the same *type* of traffic, i.e. predictable data structure accesses during each phase.
- Let $N^{\min}$ be the minimum *aggregate* requests in a cycle
- Let $N^{\max}_{\text{toi}}$ be the maximum amount of ToI requests in a cycle
- If $N^{\max}_{\text{toi}} < N^{\min}$, there is a minimum gap between each burst of ToI traffic, and arbitrarily long bursts are not possible
- Hence, this system admits $\alpha_{toi} < \alpha$.

Progressively complex models

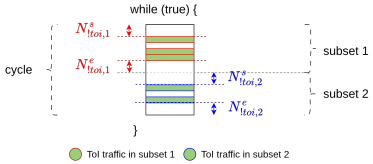
We iterated on the idea, finding improved versions which take more structural information into account



Basic

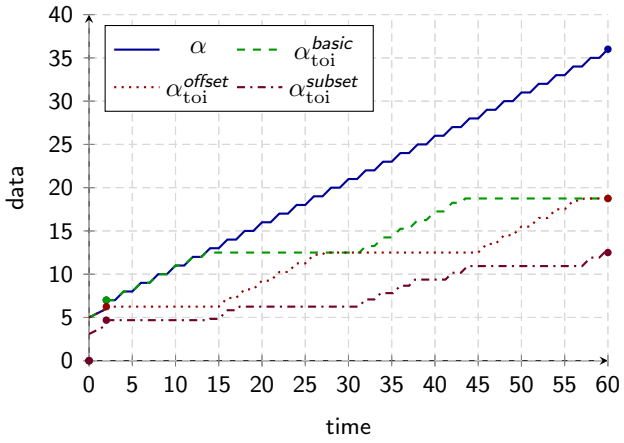


Offset-aware



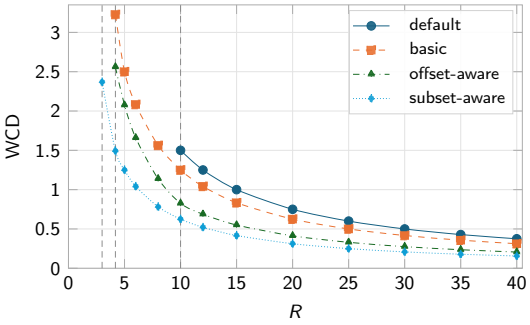
Subset-aware

Progressively complex models

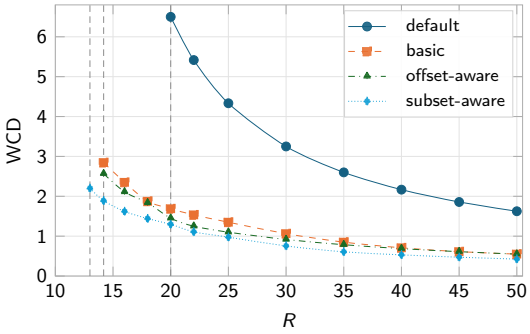


α (blue), and α_{toi} from basic (green), offset-aware (red), subset-aware (purple).

Improvements in example scenarios



Scenario A: ToI traffic served by a rate-latency node.



Scenario B: ToI is the competing flow, so our flow gets better residual service.

- All three methods widen the stability region.
- Residual-service case: subset-aware WCD up to $\sim 30\%$ smaller than the basic method's.

Towards the real scenario

- The above work finds what the mathematical model *needs* to do proper splitting, remove pessimism and improve bounds.

Towards the real scenario

- The above work finds what the mathematical model *needs* to do proper splitting, remove pessimism and improve bounds.
- The technique is general, and applies whenever there is some demultiplexing

Towards the real scenario

- The above work finds what the mathematical model *needs* to do proper splitting, remove pessimism and improve bounds.
- The technique is general, and applies whenever there is some demultiplexing
  - memory requests to different DRAM banks/chip/etc.
  - memory requests that hit the cache or not

Towards the real scenario

- The above work finds what the mathematical model *needs* to do proper splitting, remove pessimism and improve bounds.
- The technique is general, and applies whenever there is some demultiplexing
  - memory requests to different DRAM banks/chip/etc.
  - memory requests that hit the cache or not
- What's next: how do we actually derive this information from processes?

Towards the real scenario

- The above work finds what the mathematical model *needs* to do proper splitting, remove pessimism and improve bounds.
- The technique is general, and applies whenever there is some demultiplexing
  - memory requests to different DRAM banks/chip/etc.
  - memory requests that hit the cache or not
- What's next: how do we actually derive this information from processes?
- Annotated control-flow graphs with address ranges and structural annotations on accesses of each program phase

Towards the real scenario

- The above work finds what the mathematical model *needs* to do proper splitting, remove pessimism and improve bounds.
- The technique is general, and applies whenever there is some demultiplexing
  - memory requests to different DRAM banks/chip/etc.
  - memory requests that hit the cache or not
- What's next: how do we actually derive this information from processes?
- Annotated control-flow graphs with address ranges and structural annotations on accesses of each program phase
- Deriving these CFGs from binaries

Takeaways

- Worst-case memory latencies has to consider many processors and memory elements
- While load-balancing is easy to model *on average*, its *worst-case* introduces lots of pessimism
- SOTA modelling forces *duplication* of arrival curves at each demultiplexing point
- We investigated the *structural information* needed to avoid this pessimism
- Next step is deriving these parameters out of tasks, with the current direction being *annotated CFGs*

Thank you!