UCLouvain

WoSC 10 — 10th International Workshop on Serverless Computing
December 2—3, 2024 — Hong Kong

Energy-Aware Scheduling of a Serverless Workload in an ISA-Heterogeneous Cluster

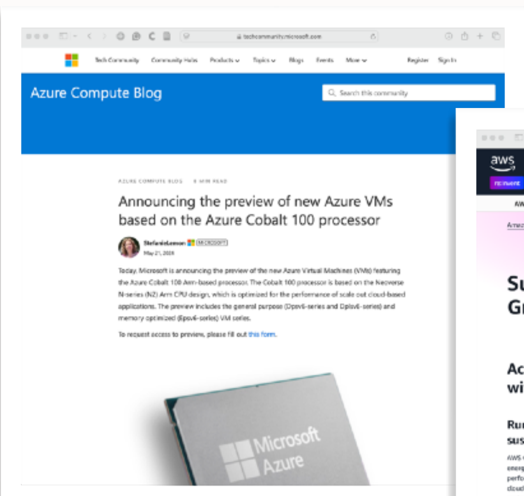
Simon Arys, Romain Carlier, and Etienne Rivière

UCLouvain, Belgium

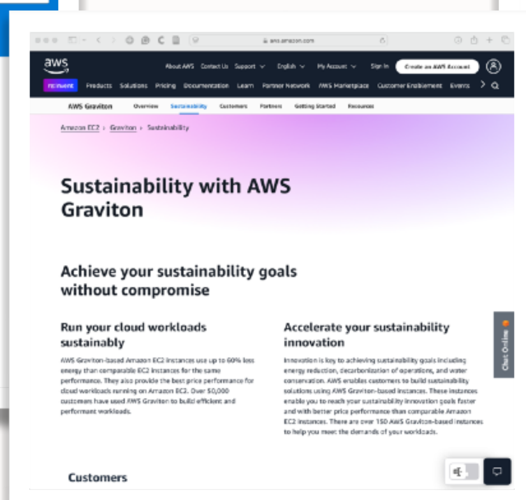
etienne.riviere@uclouvain.be

ISA heterogeneity

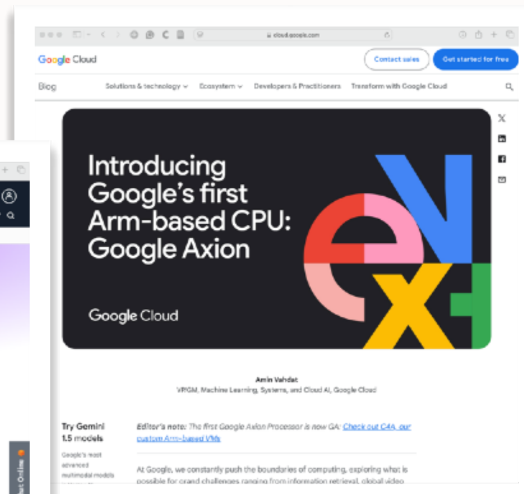
- **x86**: *de facto* standard in data centers (~80/90% of the market)
  - Intel Xeon, AMD EPYC (Zen), …
  - CISC (Complex Instruction Set)—*high performance per Hz*
- Increasingly more **ARM** CPUs
  - RISC (Reduced Instruction Set)—*high performance per Watt*
  - ARM-based cloud offerings by Amazon, Google, and Azure (announced)



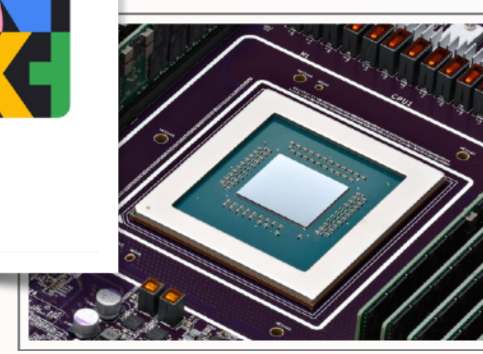
Azure — May 2024



AWS — since 2018



Google — Sep. 2024



Google's Axion ARM processors

Serverless: An opportunity

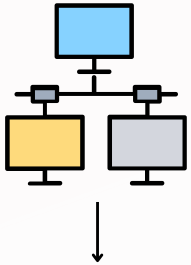
- Different workloads have different performance and energy consumption on different CPUs
  - 💡 Schedule workloads on x86 or ARM, depending on *performance* and *energy efficiency*

Pay-per-use model

Simplified Development

- Serverless model ❤️ heterogeneity
  - Decouples code from infrastructure, statelessness
  - Functions can run on different hardware interchangeably
    - Interpreted (node.js, python) — different interpreters
    - Bytecode (Java, WebAssembly) — different virtual machines
    - Native (C++, Rust) — multiple compilation targets
  - ➡ Running on ARM or x86 can be a runtime decision

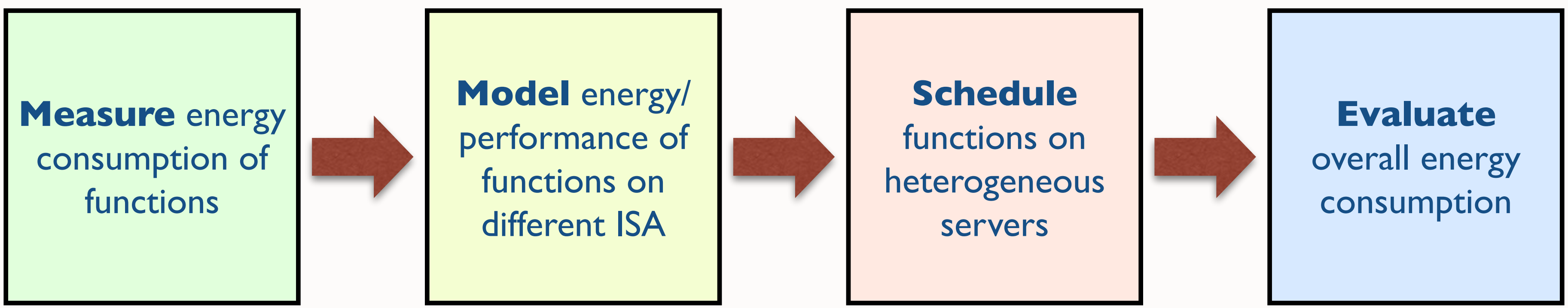
Stateless Event-driven execution



Functions can be scheduled to any machine **interchangeably**, thanks to those being **stateless** and **ISA-independent**

Our research question

- Can we schedule a serverless workload onto a heterogeneous { x86, ARM } cluster while
  - Matching performance requirements (req. / s / function)?
  - Reducing overall energy consumption (watt.h)?



WoSC 10 — Arys, Carlier, and Rivière — etienne.riviere@uclouvain.be

Our target hardware

- Cluster of two mid/high-end machines
  - Similar price point (8-10K€) mid-2022
  - **x86** machine: two Xeon CPUs (TDP 350W)
  - **ARM** machine: one Ampere Altra Max CPU (TDP 250W)
  - Same other characteristics: 256GB RAM, SSDs, 100 Gbe, …

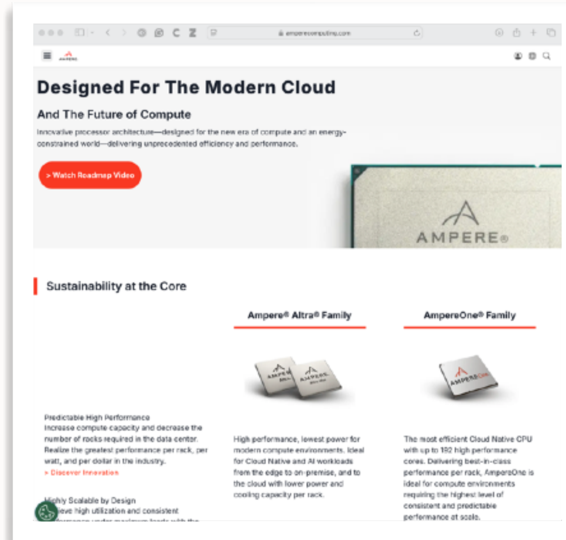
Intel
Xeon Gold 5318Y

Architecture: x86
Cores: 2 x 24 (48)
Threads: 2 x 48 (96)
TDP: 2 x 175W (350W)

(dual socket machine)

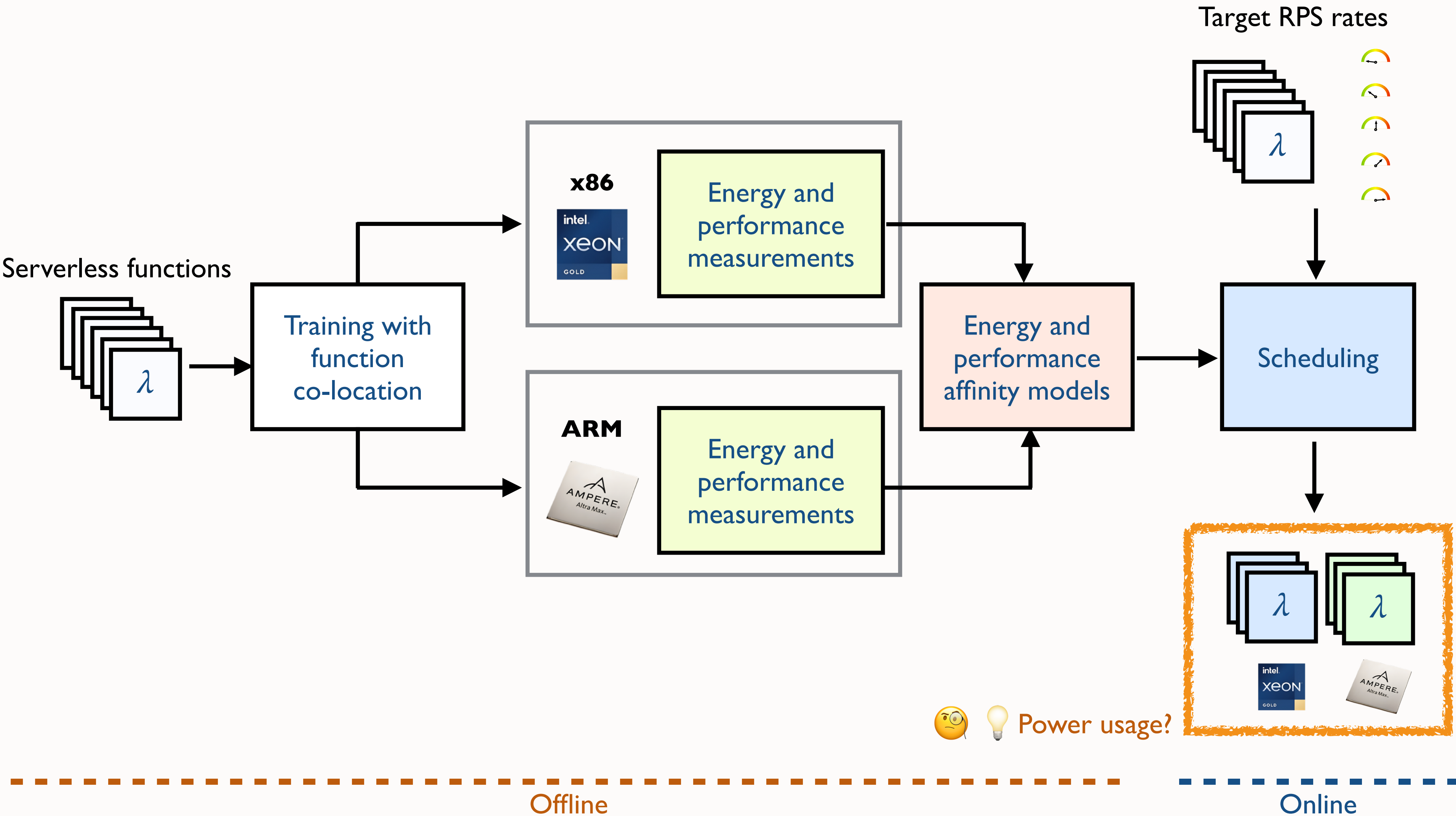
Ampere
Altra Max

Architecture: ARM
Cores: 128
Threads: 128
TDP: 250W



Ampere commercial arguments

Methodology overview

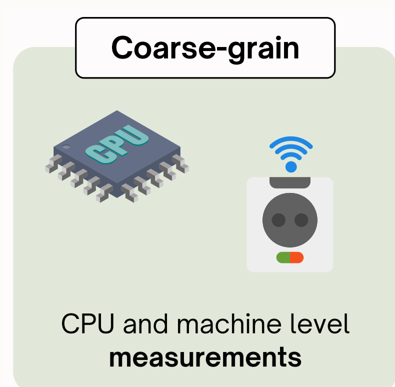


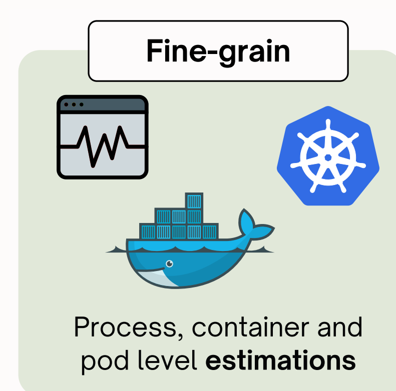
Operational assumptions

- The set of functions is known and can be profiled in advance
  - At runtime, scheduler receives target workload for each function
- Functions run as processes or containers
  - We don't assume pre-warming or keepalive policies
- High-usage scenarios
  - A single server is insufficient to match requested service rates
  - Most of the cores used to execute functions
  - High degree of function colocation

Measuring energy

- Runtime phase: global energy consumption of two servers (coarse-grain)
  - Obtained from smart PDU (Power Distribution Unit)
  - Our "ground truth" of total energy consumption
- Profiling phase: need energy consumption of individual processes (fine-grain)
  - CPU-level monitoring ignores the (software) notion of a process
  - Need to distribute measured CPU energy between processes
  - PowerAPI (https://powerapi.org/)

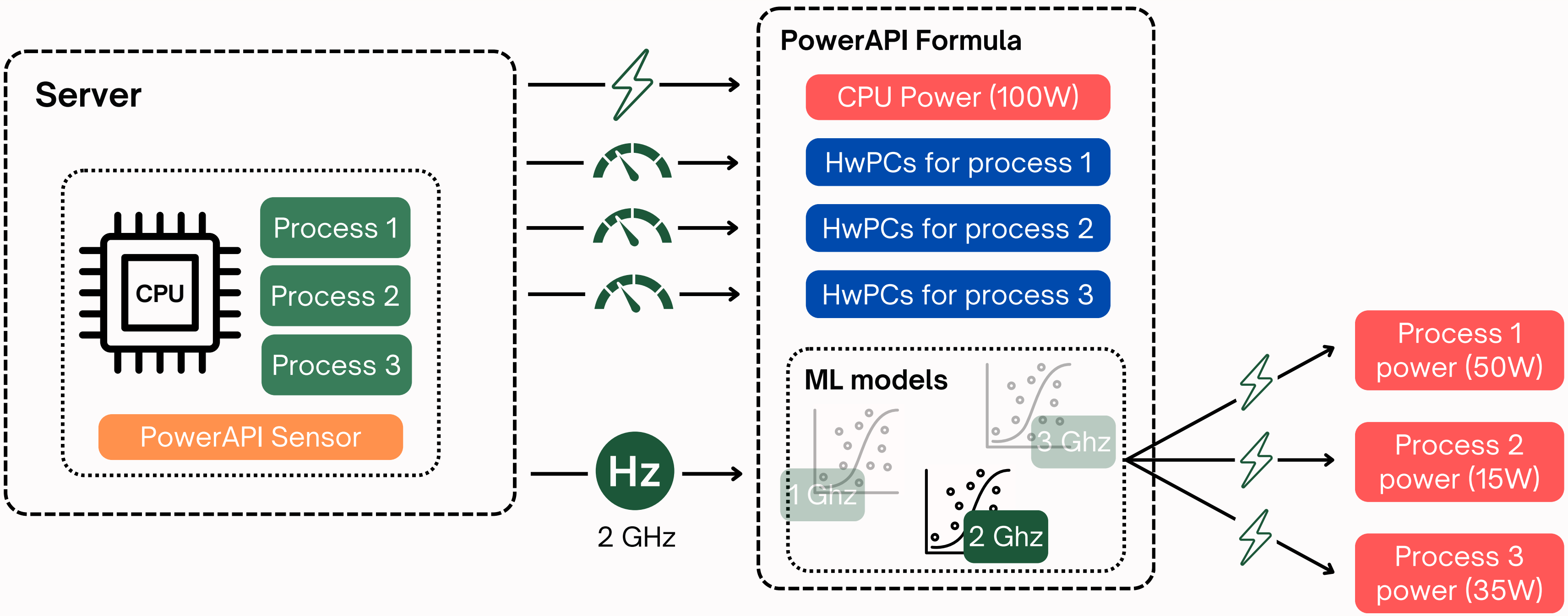




PowerAPI

PowerAPI: workflow

- PowerAPI collects multiple sensors from CPU and builds models for different frequency levels
- CPU power consumption is split between processes according to the models
- A 10+ year project led by Inria, France



HwPC = Hardware Performance Counter (ex: CPU cycles, cache misses, ...)

Our Additions to PowerAPI

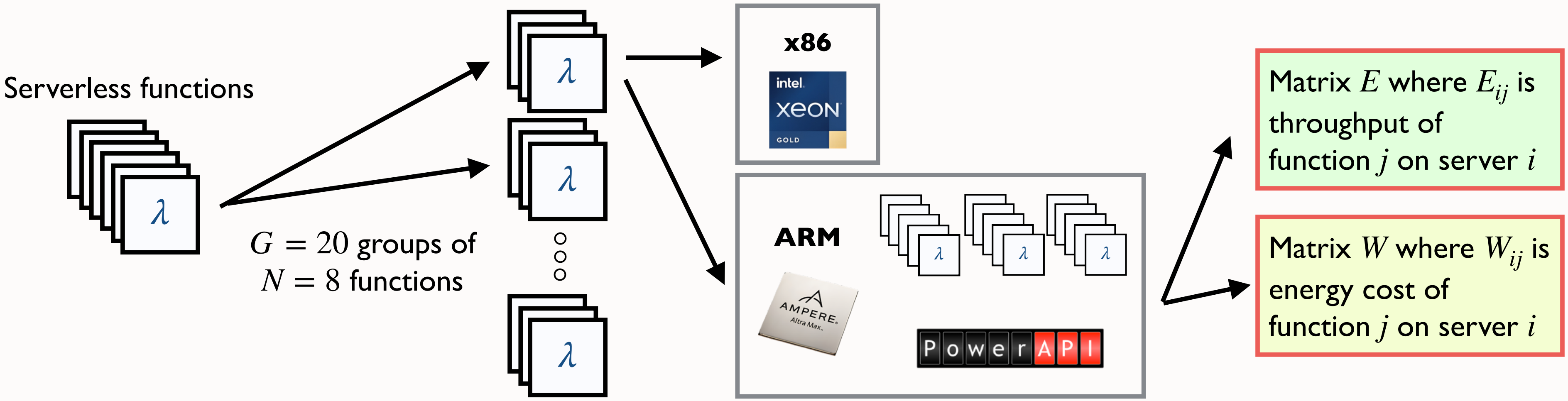
- PowerAPI focuses on x86 CPUs

- Support for Ampere's Altra Max ARM CPU
  - CPU power consumption: connect to Ampere's SMpro system control processor (*instead of RAPL on x86*)
  - Cores frequency and hardware performance counters: Collaborative Processor Performance Control (CPPC) driver in the Linux kernel (*instead of Model-Specific Registers on x86*)

- Training of PowerAPI models for the Altra Max
  - Hardware performance counters correlated with process energy consumption: CPU cycles, retired instructions, and stalled cycles

Energy- and performance affinity models

- Every function in the training set must be probed for
  - Its performance (executions/second, on one core)
  - Its energy efficiency (watt.hour/execution) using PowerAPI
- Measuring functions in isolation ignores the impact of co-location
  - Shared resources (memory bandwidth, last-level caches, etc.)
  - Testing all possible combinations is intractable
- Measure functions as G bags of N randomly chosen functions
  - Each function is represented in at least G_{min} groups ($G = 20, N = 8, G_{min} = 2$)

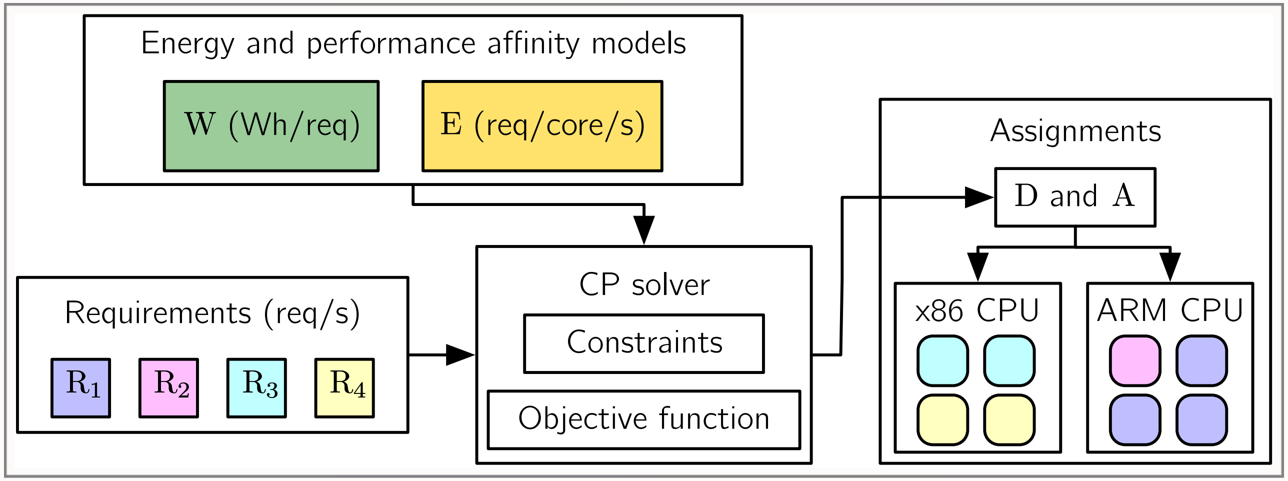


Affinity-based scheduling

- Assign cores to each function on one of the two servers
  - W and E: energy and performance models
  - R: throughput requirements, R_i is the number of requests per second that function i must support
  - C: vector of servers' capacities, C_i is the number of cores of server i
  - D: boolean matrix, $D_{ij} = \top$ if function j deployed on server i
    - Note: function j is deployed to a single server
  - A: core assignment vector, A_i is the number of cores for function i

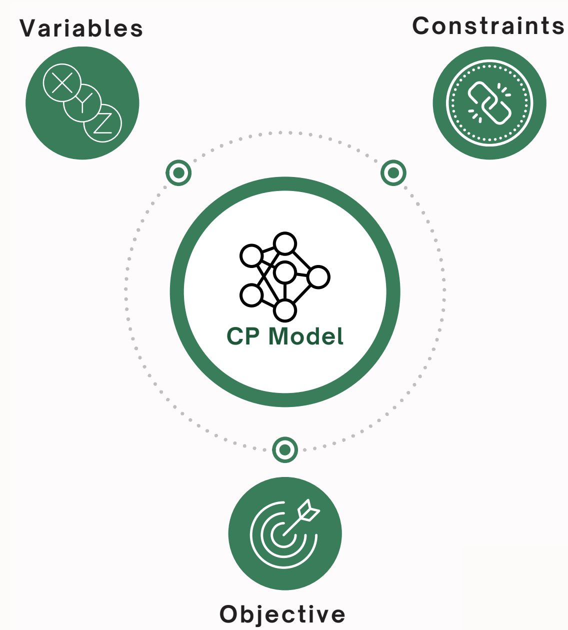
Our inputs

Our outputs



Constraint Problem

- The scheduling problem is defined as an optimization under constraints
- Constraints:
  - Each function on one server: $\sum_{i=1}^{m} (D_{ij}) = 1 \quad \forall j \in [1,n]$
  - Servers capacity: $C_i \geq \sum_{j=1}^{n} (D_{ij} \cdot A_i) \quad \forall i \in [1,m]$
- Assignment of cores to functions
  - $A_j = \left\lceil \sum_{i=1}^{m} \left(D_{ij} \cdot \frac{R_j}{E_{ij}} \right) \right\rceil \quad \forall j \in [1,n]$
- Objective functions that we need to minimize
  - Energy-aware policy $O_E : \sum_{i=1}^{m} \left(\sum_{j=1}^{n} (D_{ij} \cdot W_{ij} \cdot R_j) \right)$
  - Core-aware policy minimizes occupied cores: $O_C : \sum_{i=1}^{m} \left(\sum_{j=1}^{n} (D_{ij} \cdot A_j) \right)$
- Expressed using MiniZinc, solved using Gecode solver



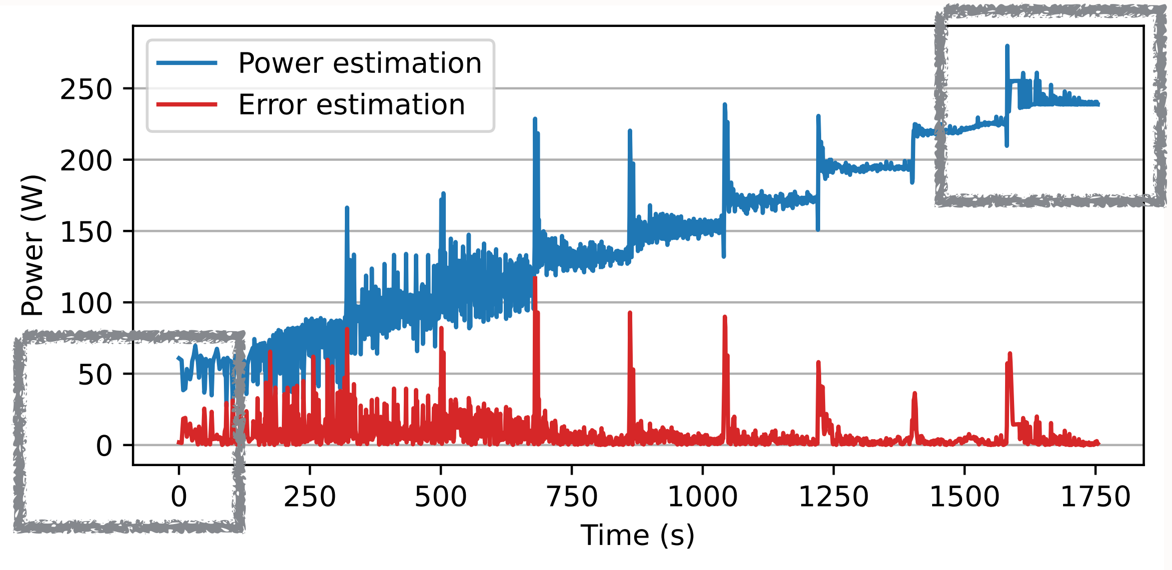
Evaluation

- Can the energy-aware policy outperform a random placement or the energy-unaware, core-aware policy?

- Four steps
  1. Validation of PowerAPI on ARM
  2. Definition of a serverless workload
  3. Results of performance and energy affinity models
  4. Impact of scheduling policies on global consumption

Validation of Ampere PowerAPI integration

- PowerAPI monitors the difference between CPU global estimation and results from the models
  - Self-calibration if over a threshold
- Validation using `stress-ng` with 0% to 100% load (all 128 cores saturated), increment every 3 minutes



Better accuracy with higher load, estimation matches TDP

Aligned with announced idle consumption of the CPU (50W)

Workload: 22 serverless functions

Cryptographic, server-side operations

4 runtime environments

- Interpreted
- JVM
- Native
- WebAssembly

4 Programming languages

- Python
- C++
- Node.js
- Java

	Function	Runtimes	Description
			Category 1: Crypto
1–2	**El Passo**	C++, WebAssembly	Compute anonymous credentials for single-sign-on.
3–5	**JWT**	Python, Node.js, Java	Create and sign JWT token.
			Category 2: Media
6–8	**Thumbnail**	Python, Node.js, Java	Resize a base64-encoded image.
9–10	**Video to GIF**	Python, Node.js	Transcode a video to a GIF using ffmpeg.
11	**Img Recognition**	Python	Object recognition in an image using the AlexNet model of Torchvision.
			Category 3: Scientific
12	**Matrix NumPy**	Python	Compute dot product of two random 30x30 matrices with NumPy.
13–15	**Matrix native**	Python, Node.js, Java	Compute dot product of two random 30x30 matrices using native code.
16	**PageRank**	Python	Compute pagerank on a 500-vertex, 250-edge graph using igraph.
			Category 4: Web
17–19	**HTML**	Python, Node.js, Java	Fill HTML template using jinja2 (Python), Mustache (Node.JS), or FreeMaker (Java).
20–22	**Zip**	Python, Node.js, Java	Compress 3 files into a single zip archive.

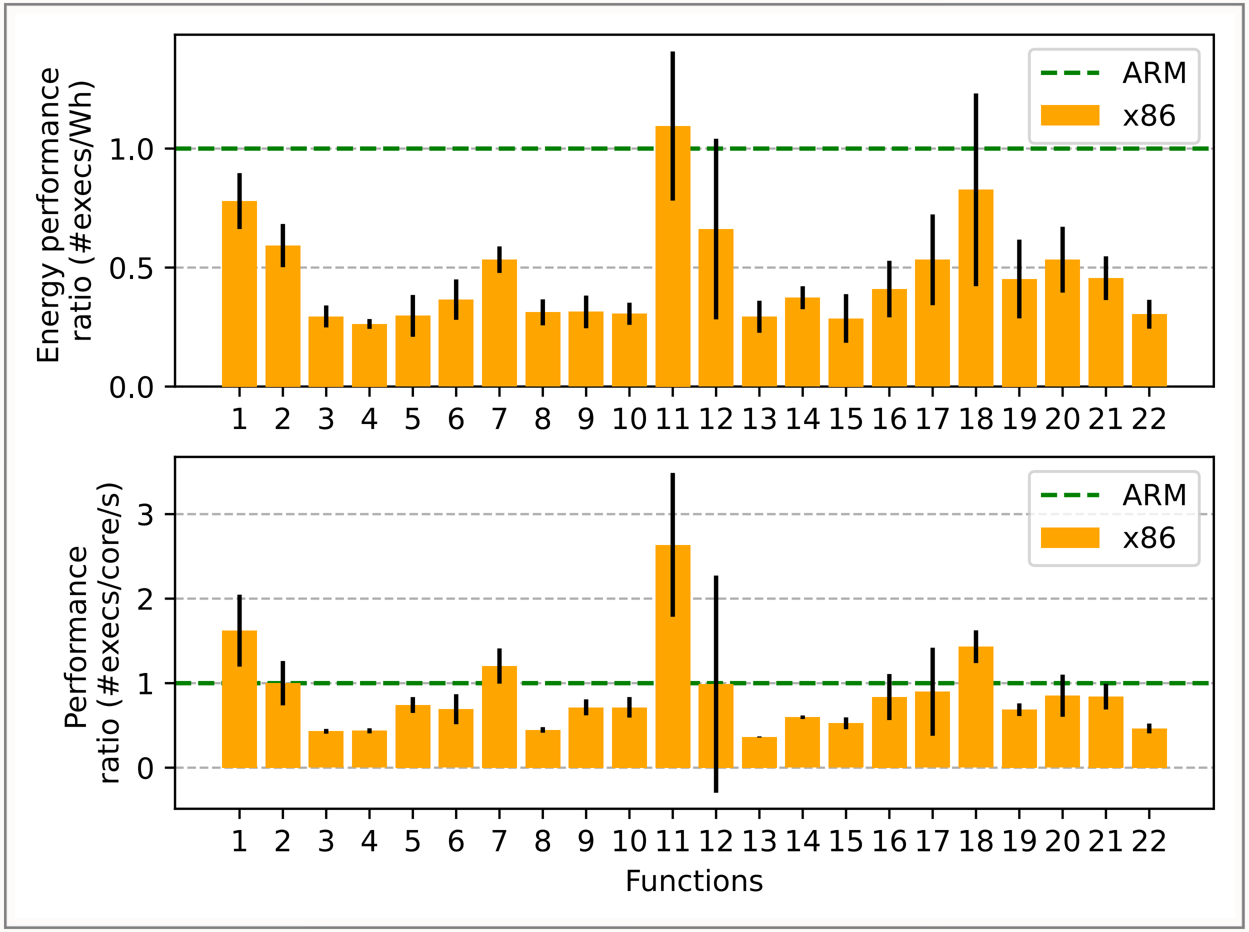
Function taken or adapted from SeBS

Functions coded for or compiled to different runtimes

Performance and energy

(normalized to ARM results)

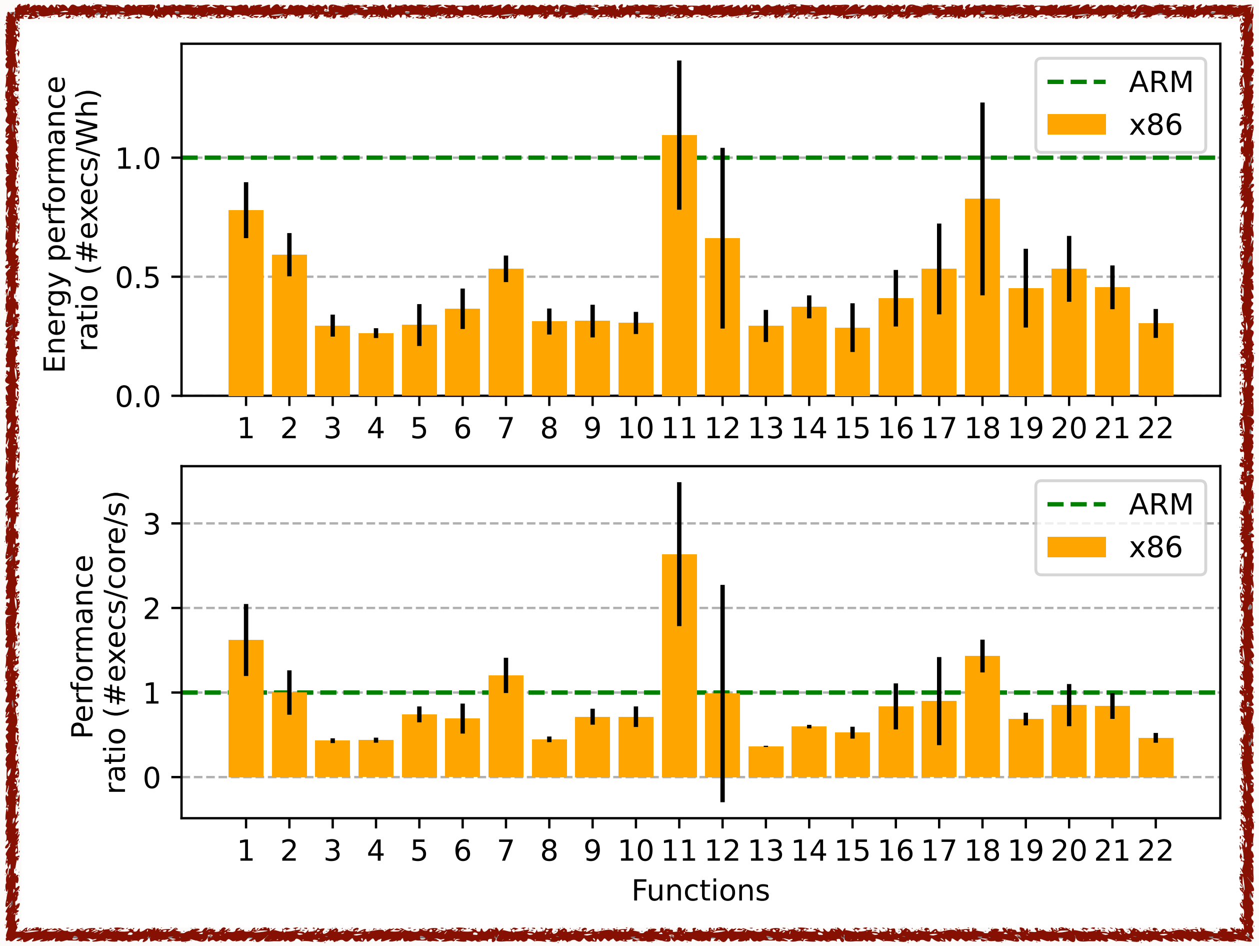
	Function	Runtimes	Description
		Category 1: Crypto	
1–2	**El Passo**	C++, WebAssembly	Compute anonymous credentials for single-sign-on [28].
3–5	**JWT**	Python, Node.js, Java	Create and sign JWT token.
		Category 2: Media	
6–8	**Thumbnail**	Python, Node.js, Java	Resize a base64-encoded image.
9–10	**Video to GIF**	Python, Node.js	Transcode a video to a GIF using ffmpeg.
11	**Img Rec.**	Python	Object recognition in an image using the AlexNet model of Torchvision.
		Category 3: Scientific	
12	**Matrix NumPy**	Python	Compute dot product of two random 30x30 matrices with NumPy.
13–15	**Matrix native**	Python, Node.js, Java	Compute dot product of two random 30x30 matrices using native code.
16	**PageRank**	Python	Compute pagerank on a 500-vertex, 250-edge graph using igraph.
		Category 4: Web	
17–19	**HTML**	Python, Node.js, Java	Fill HTML template using jinja2 (Python), Mustache (Node.JS), or FreeMaker (Java).
20–22	**Zip**	Python, Node.js, Java	Compress 3 files into a single zip archive.



Performance and energy

(normalized to ARM results)

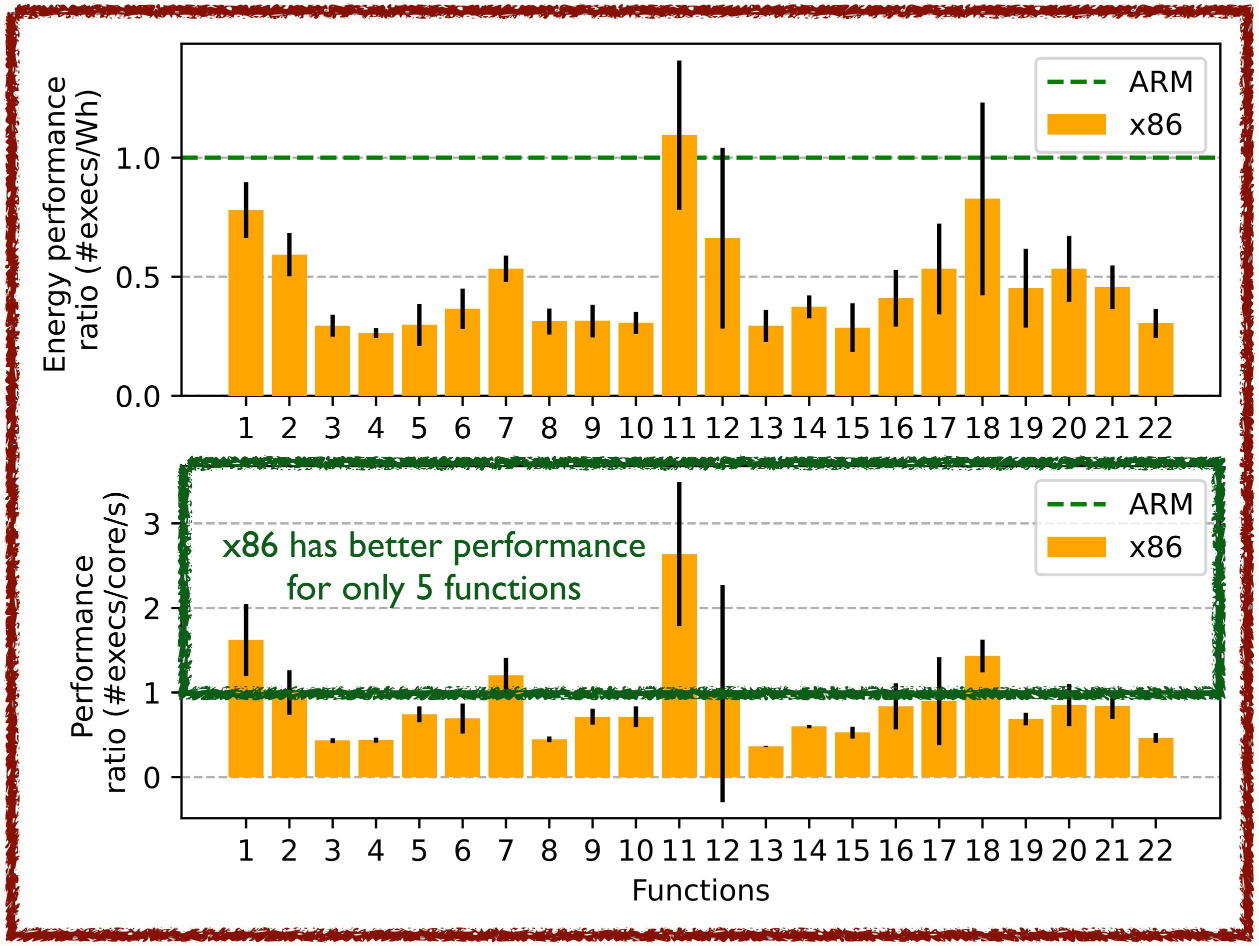
	Function	Runtimes	Description
Category 1: Crypto			
1–2	**El Passo**	C++, WebAssembly	Compute anonymous credentials for single-sign-on [28].
3–5	**JWT**	Python, Node.js, Java	Create and sign JWT token.
Category 2: Media			
6–8	**Thumbnail**	Python, Node.js, Java	Resize a base64-encoded image.
9–10	**Video to GIF**	Python, Node.js	Transcode a video to a GIF using ffmpeg.
11	**Img Rec.**	Python	Object recognition in an image using the AlexNet model of Torchvision.
Category 3: Scientific			
12	**Matrix NumPy**	Python	Compute dot product of two random 30x30 matrices with NumPy.
13–15	**Matrix native**	Python, Node.js, Java	Compute dot product of two random 30x30 matrices using native code.
16	**PageRank**	Python	Compute pagerank on a 500-vertex, 250-edge graph using igraph.
Category 4: Web			
17–19	**HTML**	Python, Node.js, Java	Fill HTML template using jinja2 (Python), Mustache (Node.JS), or FreeMaker (Java).
20–22	**Zip**	Python, Node.js, Java	Compress 3 files into a single zip archive.



21/22 functions have better performance **or** better energy efficiency on ARM

Performance and energy
(normalized to ARM results)

	Function	Runtimes	Description
Category 1: Crypto			
1–2	**El Passo**	C++, WebAssembly	Compute anonymous credentials for single-sign-on [28].
3–5	**JWT**	Python, Node.js, Java	Create and sign JWT token.
Category 2: Media			
6–8	**Thumbnail**	Python, Node.js, Java	Resize a base64-encoded image.
9–10	**Video to GIF**	Python, Node.js	Transcode a video to a GIF using ffmpeg.
11	**Img Rec.**	Python	Object recognition in an image using the AlexNet model of Torchvision.
Category 3: Scientific			
12	**Matrix NumPy**	Python	Compute dot product of two random 30x30 matrices with NumPy.
13–15	**Matrix native**	Python, Node.js, Java	Compute dot product of two random 30x30 matrices using native code.
16	**PageRank**	Python	Compute pagerank on a 500-vertex, 250-edge graph using igraph.
Category 4: Web			
17–19	**HTML**	Python, Node.js, Java	Fill HTML template using jinja2 (Python), Mustache (Node.JS), or FreeMaker (Java).
20–22	**Zip**	Python, Node.js, Java	Compress 3 files into a single zip archive.

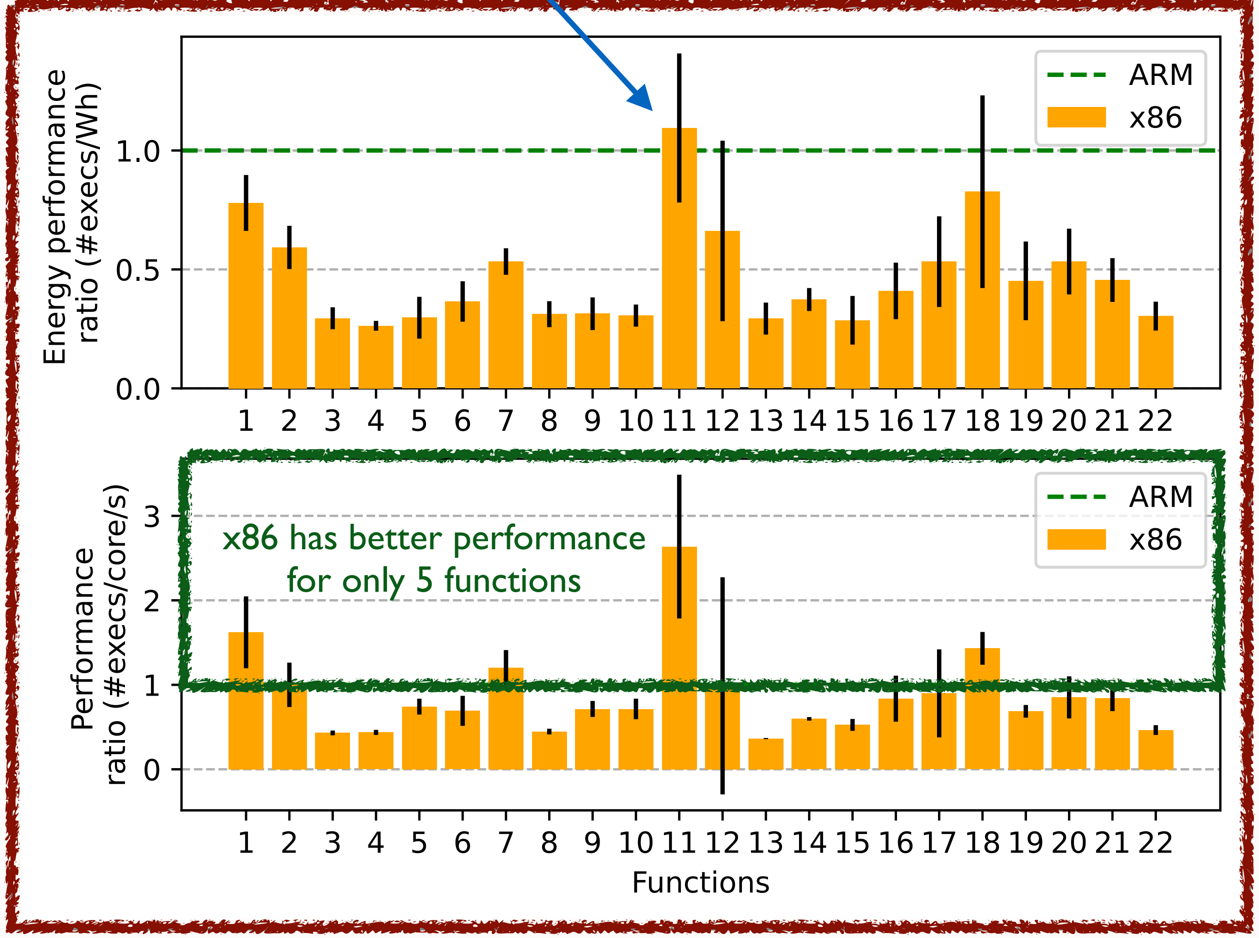


21/22 functions have better performance **or** better energy efficiency on ARM

Performance and energy
(normalized to ARM results)

One function has better energy and performance results on x86

Function		Runtimes	Description
Category 1: Crypto			
1–2	**El Passo**	C++, WebAssembly	Compute anonymous credentials for single-sign-on [28].
3–5	**JWT**	Python, Node.js, Java	Create and sign JWT token.
Category 2: Media			
6–8	**Thumbnail**	Python, Node.js, Java	Resize a base64-encoded image.
9–10	**Video to GIF**	Python, Node.js	Transcode a video to a GIF using ffmpeg.
11	**Img Rec.**	Python	Object recognition in an image using the AlexNet model of Torchvision.
Category 3: Scientific			
12	**Matrix NumPy**	Python	Compute dot product of two random 30x30 matrices with NumPy.
13–15	**Matrix native**	Python, Node.js, Java	Compute dot product of two random 30x30 matrices using native code.
16	**PageRank**	Python	Compute pagerank on a 500-vertex, 250-edge graph using igraph.
Category 4: Web			
17–19	**HTML**	Python, Node.js, Java	Fill HTML template using jinja2 (Python), Mustache (Node.JS), or FreeMaker (Java).
20–22	**Zip**	Python, Node.js, Java	Compress 3 files into a single zip archive.



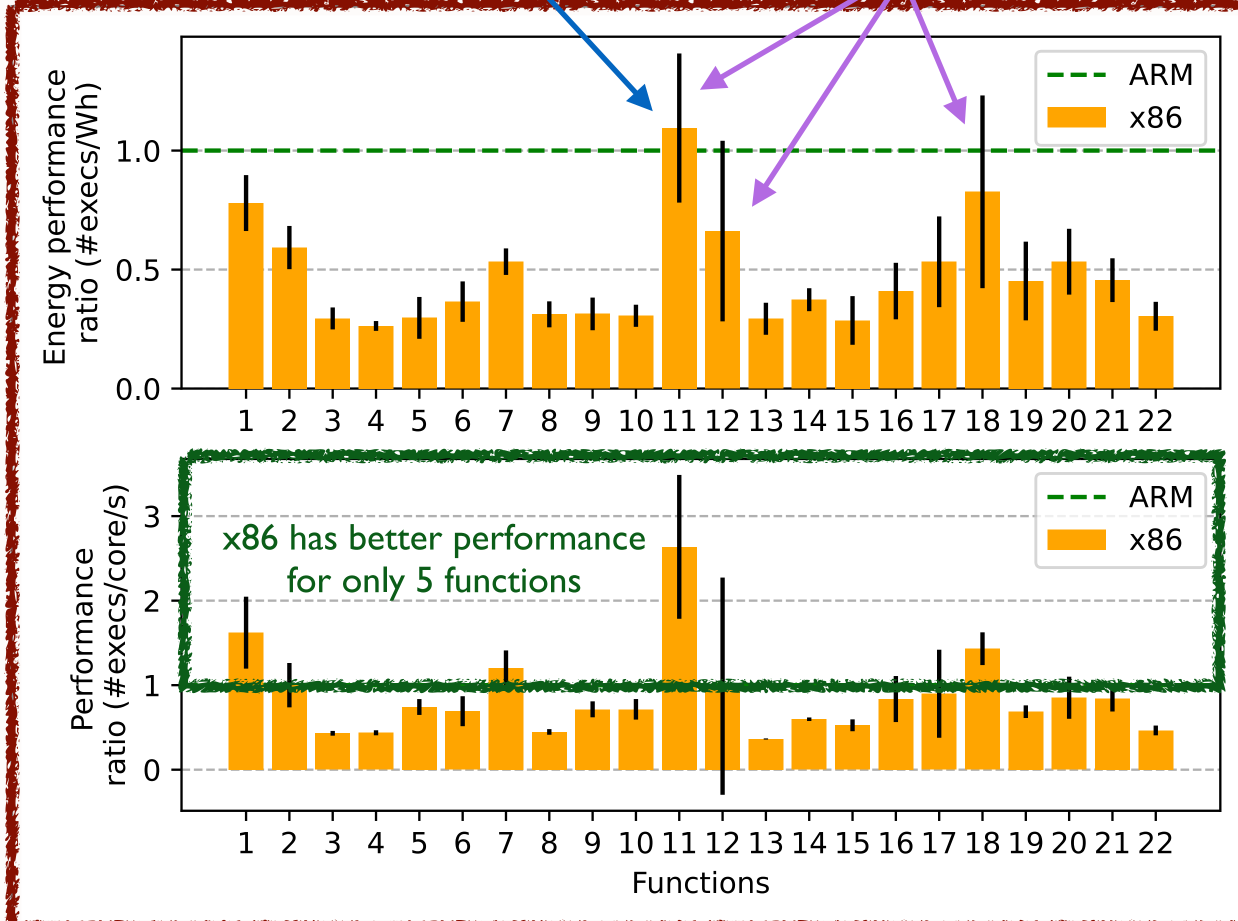
21/22 functions have better performance **or** better energy efficiency on ARM

Performance and energy
(normalized to ARM results)

	Function	Runtimes	Description
		Category 1: Crypto	
1–2	**El Passo**	C++, WebAssembly	Compute anonymous credentials for single-sign-on [28].
3–5	**JWT**	Python, Node.js, Java	Create and sign JWT token.
		Category 2: Media	
6–8	**Thumbnail**	Python, Node.js, Java	Resize a base64-encoded image.
9–10	**Video to GIF**	Python, Node.js	Transcode a video to a GIF using ffmpeg.
11	**Img Rec.**	Python	Object recognition in an image using the AlexNet model of Torchvision.
		Category 3: Scientific	
12	**Matrix NumPy**	Python	Compute dot product of two random 30x30 matrices with NumPy.
13–15	**Matrix native**	Python, Node.js, Java	Compute dot product of two random 30x30 matrices using native code.
16	**PageRank**	Python	Compute pagerank on a 500-vertex, 250-edge graph using igraph.
		Category 4: Web	
17–19	**HTML**	Python, Node.js, Java	Fill HTML template using jinja2 (Python), Mustache (Node.JS), or FreeMaker (Java).
20–22	**Zip**	Python, Node.js, Java	Compress 3 files into a single zip archive.

One function has better energy and performance results on x86

Several functions show important fluctuations due to colocation

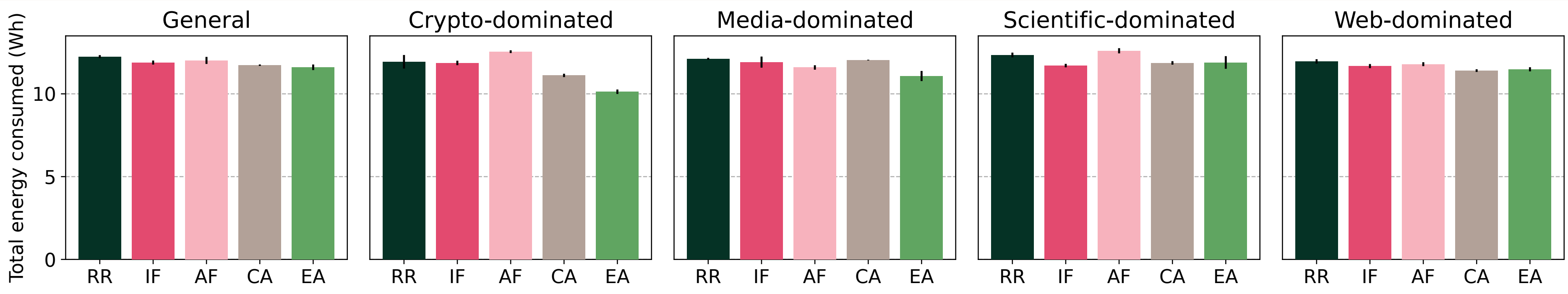


21/22 functions have better performance **or** better energy efficiency on ARM

Scheduling: results

- Complete energy consumption (server level)
  - Measured at the power distribution unit
  - CPU is one element of a whole!
- Five different workloads
  - General: 25% load (cores) per category; x-dominated: 80% of the load for category x, 20% for the rest
  - Within each category, balance the expected required number of cores per function
- All runs successfully match the required number of calls per second to each function

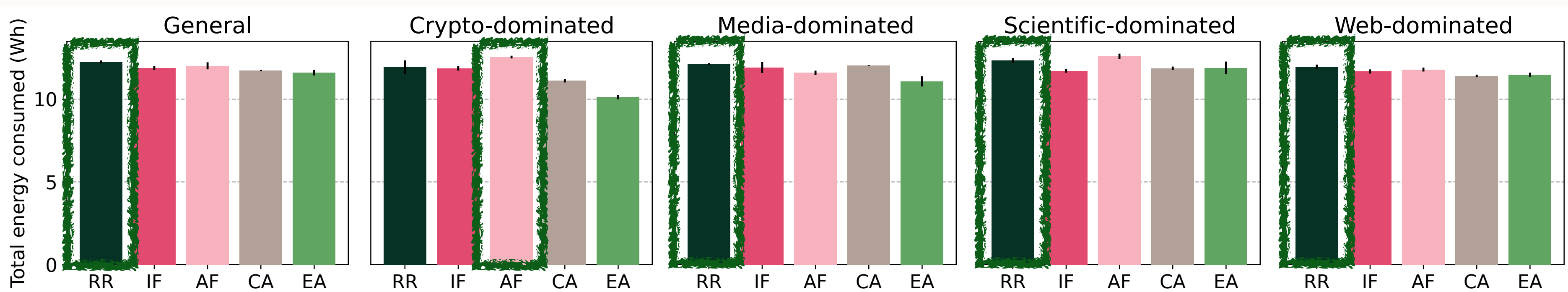
	Policy	Detail
RR	Round-Robin	Alternate placement on the two servers
IF	Intel (x86) First	First fill the x86 server, then use ARM
AF	ARM First	First fill the ARM server, then use x86
CA	Core-Aware	Baseline optimization, minimizes #cores
EA	Energy-Aware	Minimize expected energy consumption



Scheduling: results

- Complete energy consumption (server level)
  - Measured at the power distribution unit
  - CPU is one element of a whole!
- Five different workloads
  - General: 25% load (cores) per category; x-dominated: 80% of the load for category x, 20% for the rest
  - Within each category, balance the expected required number of cores per function
- All runs successfully match the required number of calls per second to each function

	Policy	Detail
RR	Round-Robin	Alternate placement on the two servers
IF	Intel (x86) First	First fill the x86 server, then use ARM
AF	ARM First	First fill the ARM server, then use x86
CA	Core-Aware	Baseline optimization, minimizes #cores
EA	Energy-Aware	Minimize expected energy consumption

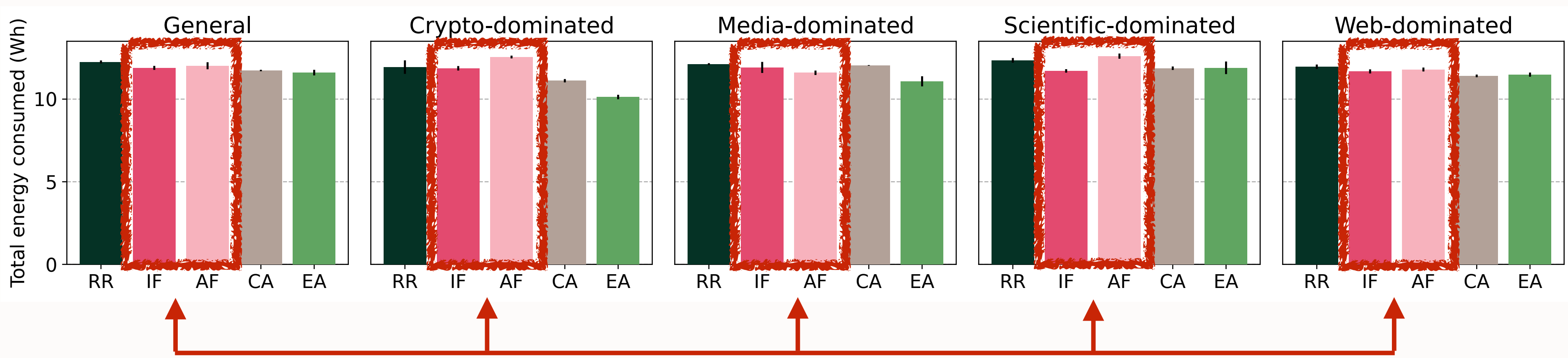


RR gives the worst results for all but Crypto-Dominated, where AF is worse

Scheduling: results

- Complete energy consumption (server level)
  - Measured at the power distribution unit
  - CPU is one element of a whole!
- Five different workloads
  - General: 25% load (cores) per category; x-dominated: 80% of the load for category x, 20% for the rest
  - Within each category, balance the expected required number of cores per function
- All runs successfully match the required number of calls per second to each function

	Policy	Detail
RR	Round-Robin	Alternate placement on the two servers
IF	Intel (x86) First	First fill the x86 server, then use ARM
AF	ARM First	First fill the ARM server, then use x86
CA	Core-Aware	Baseline optimization, minimizes #cores
EA	Energy-Aware	Minimize expected energy consumption

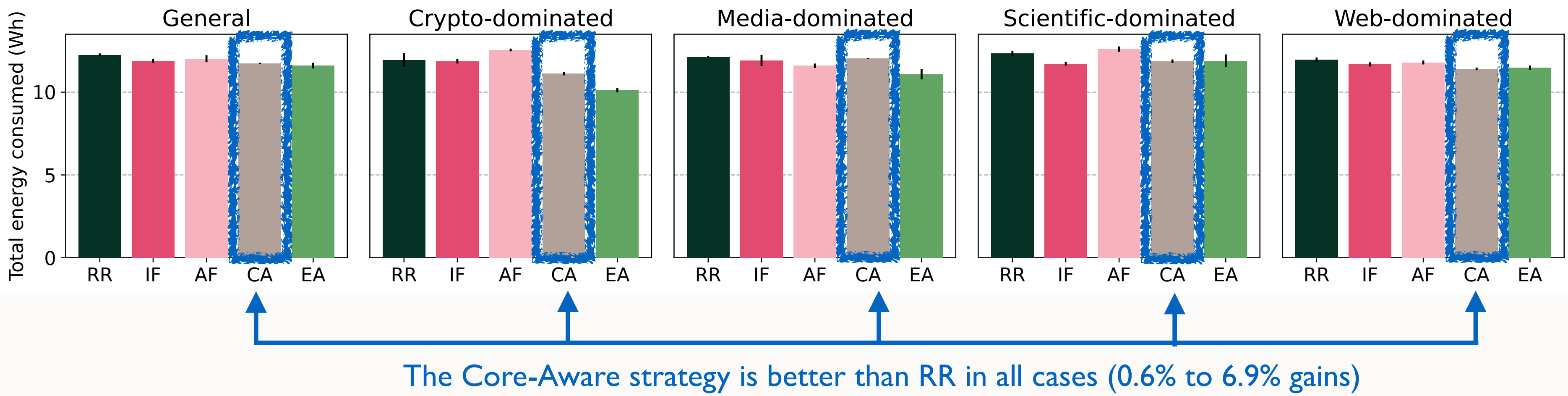


Choosing to fill one server first, even when using ARM-first, does not produce consistent gains

Scheduling: results

- Complete energy consumption (server level)
  - Measured at the power distribution unit
  - CPU is one element of a whole!
- Five different workloads
  - General: 25% load (cores) per category; x-dominated: 80% of the load for category x, 20% for the rest
  - Within each category, balance the expected required number of cores per function
- All runs successfully match the required number of calls per second to each function

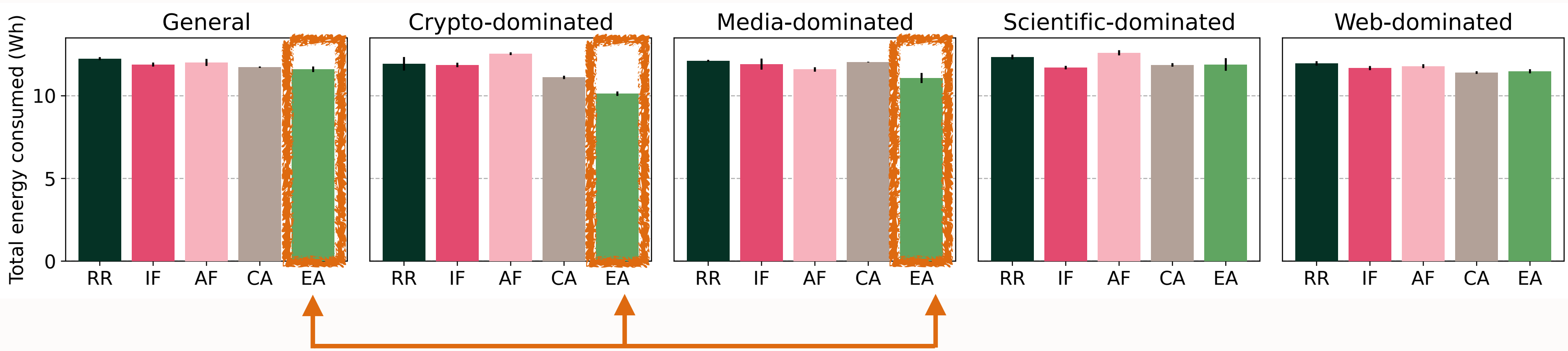
	Policy	Detail
RR	Round-Robin	Alternate placement on the two servers
IF	Intel (x86) First	First fill the x86 server, then use ARM
AF	ARM First	First fill the ARM server, then use x86
CA	Core-Aware	Baseline optimization, minimizes #cores
EA	Energy-Aware	Minimize expected energy consumption



Scheduling: results

- Complete energy consumption (server level)
  - Measured at the power distribution unit
  - CPU is one element of a whole!
- Five different workloads
  - General: 25% load (cores) per category; x-dominated: 80% of the load for category x, 20% for the rest
  - Within each category, balance the expected required number of cores per function
- All runs successfully match the required number of calls per second to each function

	Policy	Detail
RR	Round-Robin	Alternate placement on the two servers
IF	Intel (x86) First	First fill the x86 server, then use ARM
AF	ARM First	First fill the ARM server, then use x86
CA	Core-Aware	Baseline optimization, minimizes #cores
EA	Energy-Aware	Minimize expected energy consumption

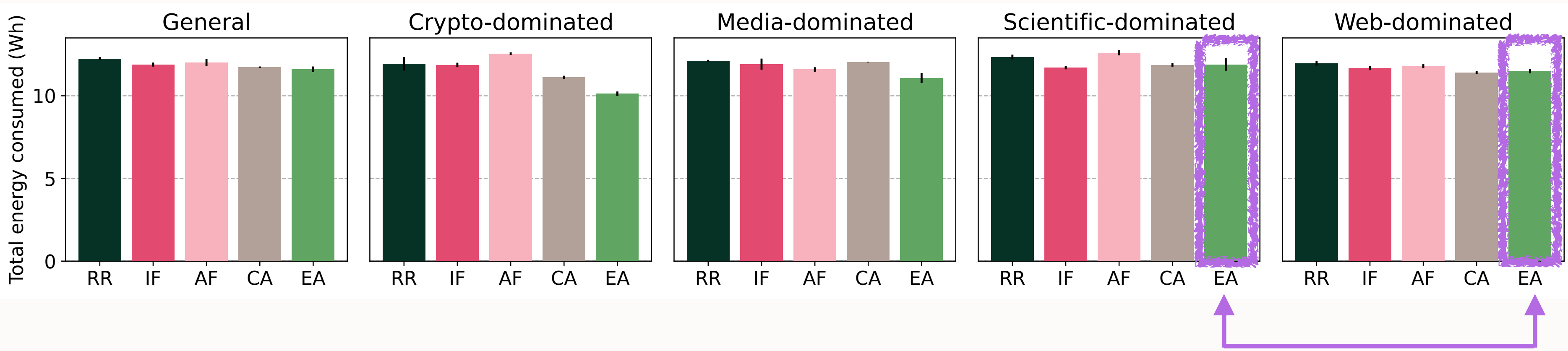


The Energy-Aware strategy gets from 5.3% to 15.2% energy reduction from RR, and improves upon CA

Scheduling: results

- Complete energy consumption (server level)
  - Measured at the power distribution unit
  - CPU is one element of a whole!
- Five different workloads
  - General: 25% load (cores) per category; x-dominated: 80% of the load for category x, 20% for the rest
  - Within each category, balance the expected required number of cores per function
- All runs successfully match the required number of calls per second to each function

	Policy	Detail
RR	Round-Robin	Alternate placement on the two servers
IF	Intel (x86) First	First fill the x86 server, then use ARM
AF	ARM First	First fill the ARM server, then use x86
CA	Core-Aware	Baseline optimization, minimizes #cores
EA	Energy-Aware	Minimize expected energy consumption



For Scientific/Web-dominated workload, no gain over CA as all ARM cores are already assigned

Conclusion

- Serverless workloads can be scheduled with better energy efficiency when considering the energy affinity of functions
  - ARM's better energy efficiency promise true for most, not all functions
  - Our servers (traditional 1U, always-on DRAM/SSD/HDD/NICs) are not necessarily the most energy-efficient; energy-aware scheduling is a small piece of a larger puzzle
- Perspectives and future work abound
  - Better models/studies of function co-location and resource contention
  - CP-based scheduling has limited scalability: hierarchical and heuristic approaches
  - Online characterization (rather than offline); integration in a serverless platform function lifecycle
  - Impact of keepalive/pre-warming policies on energy consumption

WoSC 10 — 10th International Workshop on Serverless Computing
December 2—3, 2024 — Hong Kong

Energy-Aware Scheduling of a Serverless Workload in an ISA-Heterogeneous Cluster

Thanks for your attention! Any question?

Simon Arys, Romain Carlier, and Etienne Rivière

UCLouvain, Belgium

etienne.riviere@uclouvain.be