Comb to Pipeline: Fast Software Encryption Revisited

Andrey Bogdanov Martin M. Lauridsen Elmar Tischhauser

DTU Compute, Technical University of Denmark

FSE 2015, Istanbul
March 9, 2015

The Advanced Encryption Standard (AES)

The AES

- 128-bit block cipher
- Designed by Daemen and Rijmen (1997)
- Standardized by NIST in 2001
- Now ubiquitous

The Advanced Encryption Standard (AES)

The AES

- 128-bit block cipher
- Designed by Daemen and Rijmen (1997)
- Standardized by NIST in 2001
- Now ubiquitous

Applications of the AES

- Modes of operation
- Block cipher based MACs
- Block cipher based hash functions
- Authenticated Encryption (AE) modes of operation
- ...

Timeline: AES Software Implementations

2001 Brian Gladman: Table-based, $\approx$ **25 cpb**

2007 Matsui/Nakajima: **Bitslicing**, **9.2 cpb** + overhead

2008 Bernstein/Schwabe: Table-based + micro-optimizations, **10.5 cpb** (Gladman's code now at $\approx$ 16 cpb)

Timeline: AES Software Implementations

2001 Brian Gladman: Table-based, $\approx$ **25 cpb**

2007 Matsui/Nakajima: **Bitslicing**, **9.2 cpb** + overhead

2008 Bernstein/Schwabe: Table-based + micro-optimizations, **10.5 cpb** (Gladman's code now at $\approx$ 16 cpb)

2008 Intel's first proposal for AES-NI

Timeline: AES Software Implementations

2001 Brian Gladman: Table-based, $\approx$ **25 cpb**

2007 Matsui/Nakajima: **Bitslicing**, **9.2 cpb** + overhead

2008 Bernstein/Schwabe: Table-based + micro-optimizations, **10.5 cpb** (Gladman's code now at $\approx$ 16 cpb)

2008 Intel's first proposal for AES-NI

2009 Käsper/Schwabe: Optimized bitslicing, **7 cpb**, no overhead

Timeline: AES Software Implementations

2001 Brian Gladman: Table-based, $\approx$ **25 cpb**

2007 Matsui/Nakajima: **Bitslicing**, **9.2 cpb** + overhead

2008 Bernstein/Schwabe: Table-based + micro-optimizations, **10.5 cpb** (Gladman's code now at $\approx$ 16 cpb)

2008 Intel's first proposal for AES-NI

2009 Käsper/Schwabe: Optimized bitslicing, **7 cpb**, no overhead

2010 Westmere microarchitecture with **AES-NI**, **1.25 cpb**

2011 Sandy Bridge microarchitecture: +AVX, **0.64 cpb**

2012 Ivy Bridge microarchitecture, 0.64 cpb

2013 Haswell microarchitecture: +AVX2, **0.625 cpb**

New Platforms, New Issues

With AES-NI: Performance **deficit**

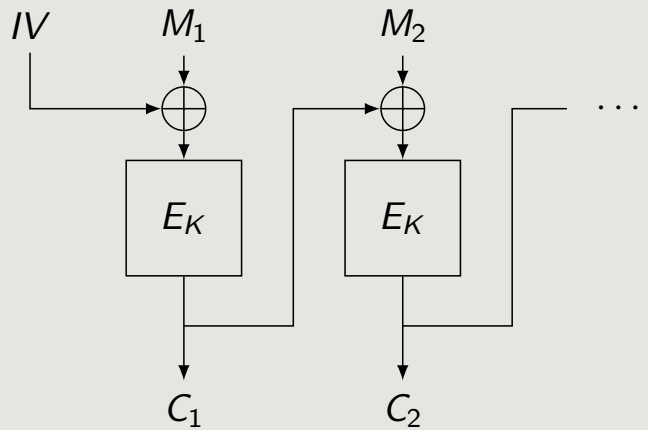
- − Inherently serial modes (e.g. CBC, CFB, OFB): $\approx$ **4.5 cpb**
- \+ Parallelizable modes (e.g. ECB, CTR): $\approx$ **0.6 – 0.7 cpb**

New Platforms, New Issues

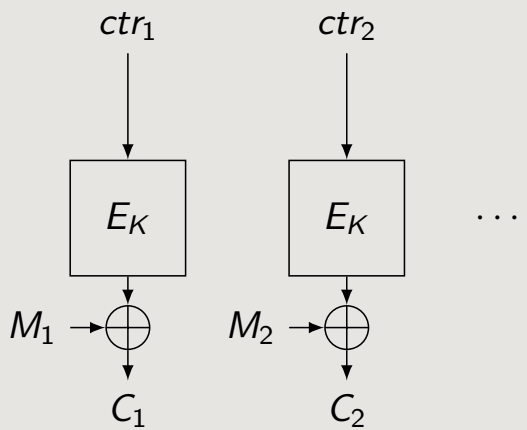
With AES-NI: Performance **deficit**

- – Inherently serial modes (e.g. CBC, CFB, OFB): $\approx$ **4.5 cpb**
- \+ Parallelizable modes (e.g. ECB, CTR): $\approx$ **0.6 – 0.7 cpb**

CBC vs. CTR



CBC encryption



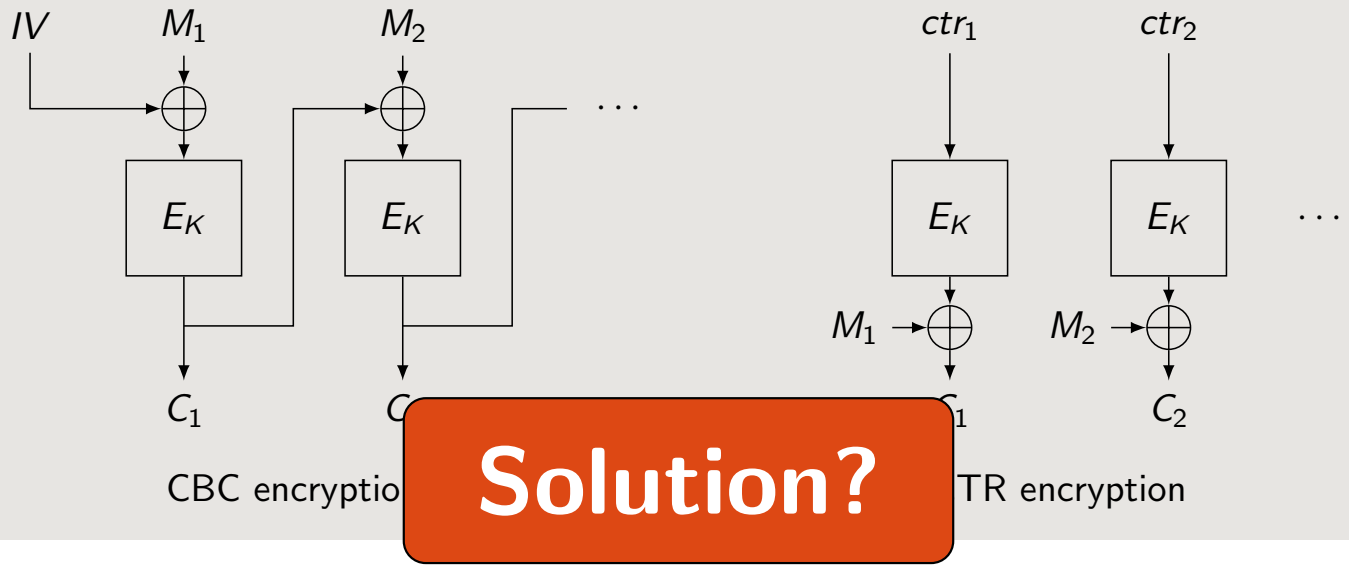
CTR encryption

New Platforms, New Issues

With AES-NI: Performance **deficit**

- – Inherently serial modes (e.g. CBC, CFB, OFB): $\approx$ **4.5 cpb**
- \+ Parallelizable modes (e.g. ECB, CTR): $\approx$ **0.6 – 0.7 cpb**

CBC vs. CTR



Outline

1. AES-NI and Pipelining
2. The Real World: Internet Traffic
3. Comb Scheduling
4. Performance Data

The AES-NI instruction set

AES-NI instructions

Encryption/decryption

- `aesenc`
- `aesdec`
- `aesenclast`
- `aesdeclast`

Key scheduling

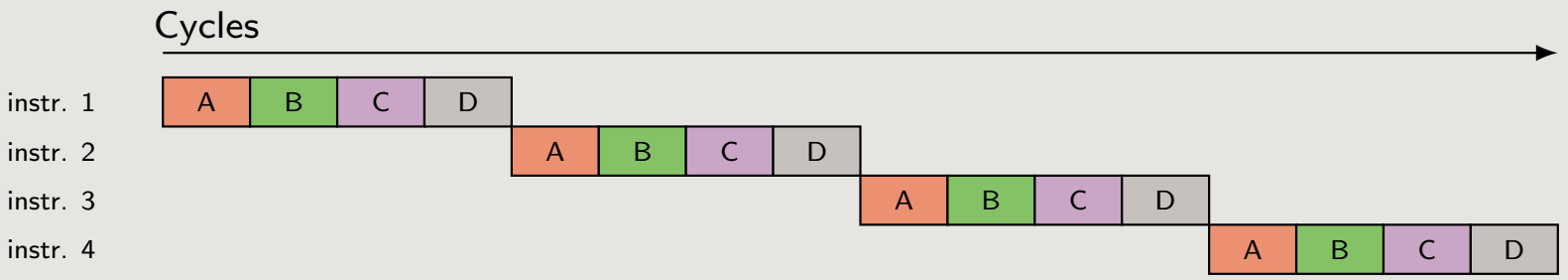
- `aesimc`
- `aeskeygenassist`

Also very usuful (e.g. in GCM):

- `pclmulqdq`: Carry-less multiplication $\{0,1\}^{64} \times \{0,1\}^{64} \to \{0,1\}^{128}$

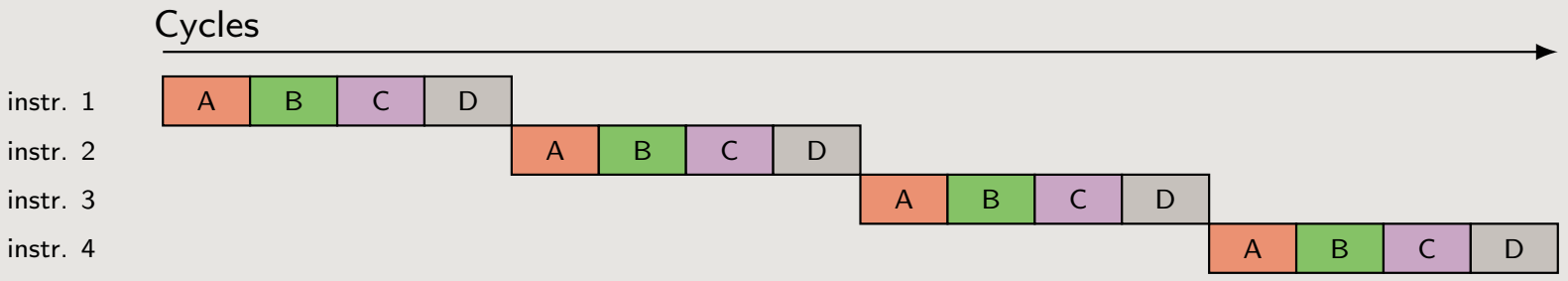
Pipelining: Exploiting Instruction-Level Parallelism

Serial execution

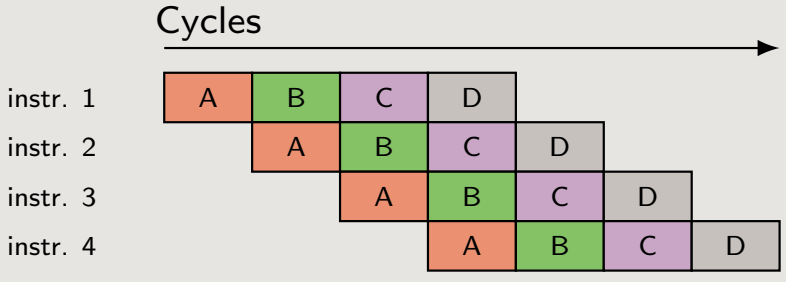


Pipelining: Exploiting Instruction-Level Parallelism

Serial execution



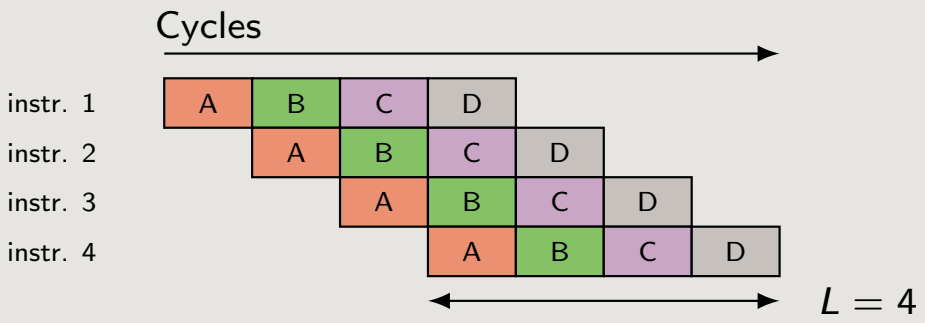
Pipelined execution



- Only applicable to **independent** instructions!

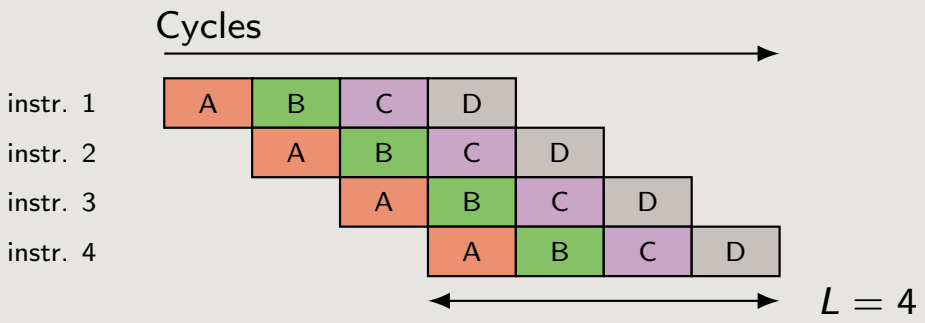
Two Important Parameters: Latency and Throughput

Latency L [cycles]

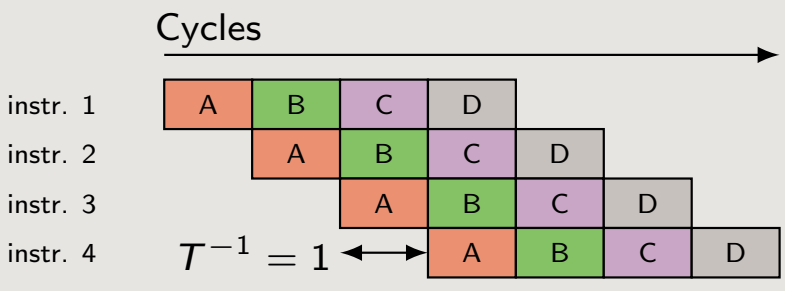


Two Important Parameters: Latency and Throughput

Latency L [cycles]



Inverse throughput T^{-1} [cycles/instruction]



Pipelined AES-NI

AES-NI on Haswell microarchitecture

Instruction	L	T^{-1}	Instruction	L	T^{-1}
`aesenc`	7	1	`aesimc`	14	2
`aesdec`	7	1	`aeskeygenassist`	10	8
`aesenclast`	7	1	`pclmulqdq`	7	2
`aesdeclast`	7	1			

Pipelined AES-NI

AES-NI on Haswell microarchitecture

Instruction	L	T^{-1}	Instruction	L	T^{-1}
`aesenc`	7	1	`aesimc`	14	2
`aesdec`	7	1	`aeskeygenassist`	10	8
`aesenclast`	7	1	`pclmulqdq`	7	2
`aesdeclast`	7	1			

- Serial code **severely penalized** on AES-NI platforms!

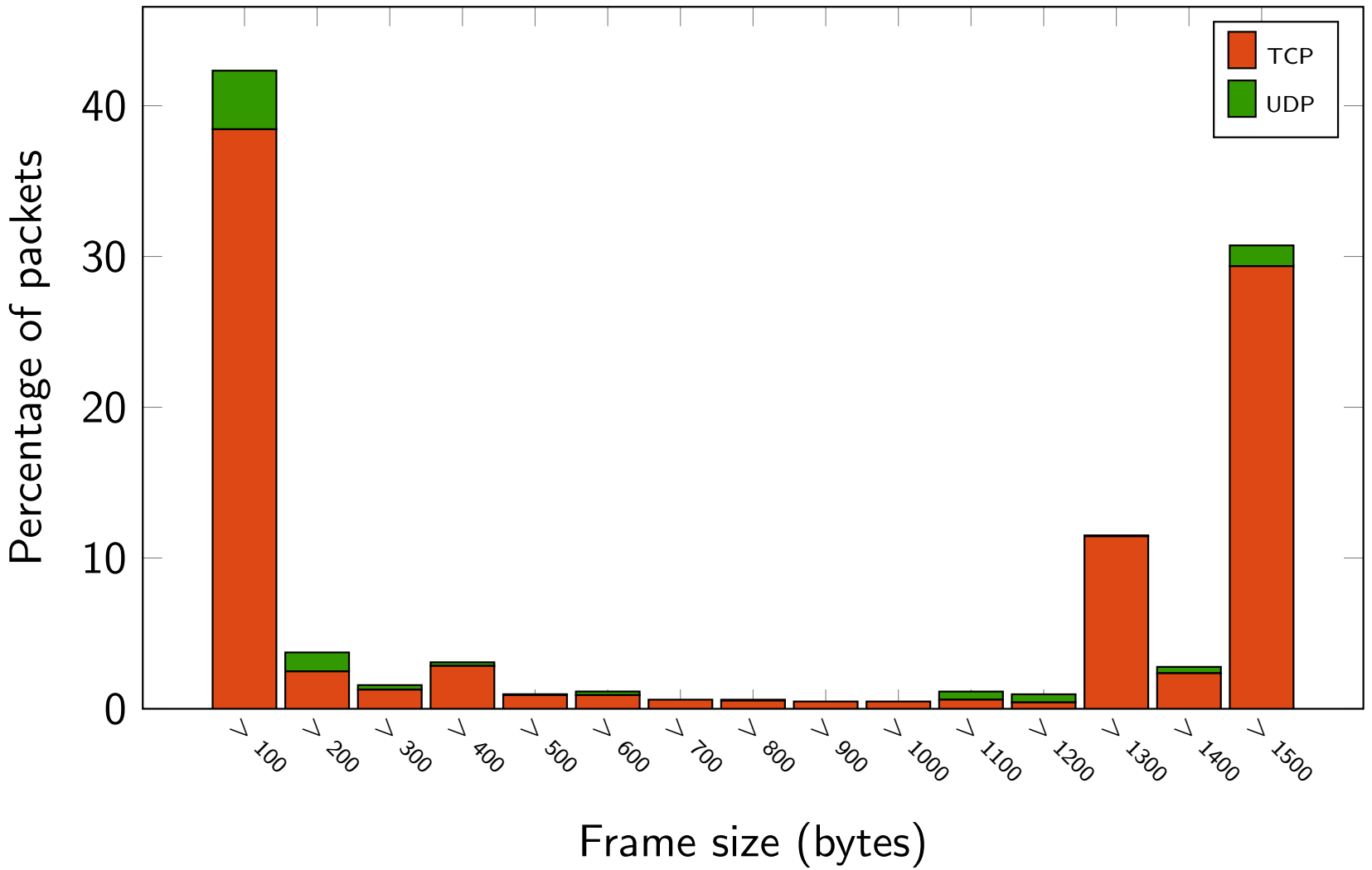
Pipelined AES-NI

AES-NI on Haswell microarchitecture

Instruction	L	T^{-1}	Instruction	L	T^{-1}
`aesenc`	7	1	`aesimc`	14	2
`aesdec`	7	1	`aeskeygenassist`	10	8
`aesenclast`	7	1	`pclmulqdq`	7	2
`aesdeclast`	7	1			

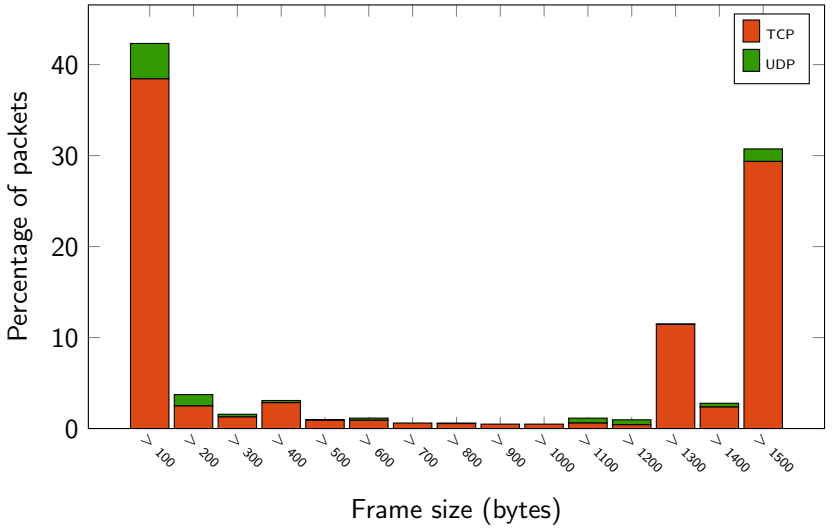
- Serial code **severely penalized** on AES-NI platforms!
- We look elsewhere for **independent data** for serial modes

Reality Check: Typical Internet Traffic



Internet packet size distribution 2012 [MK2012]

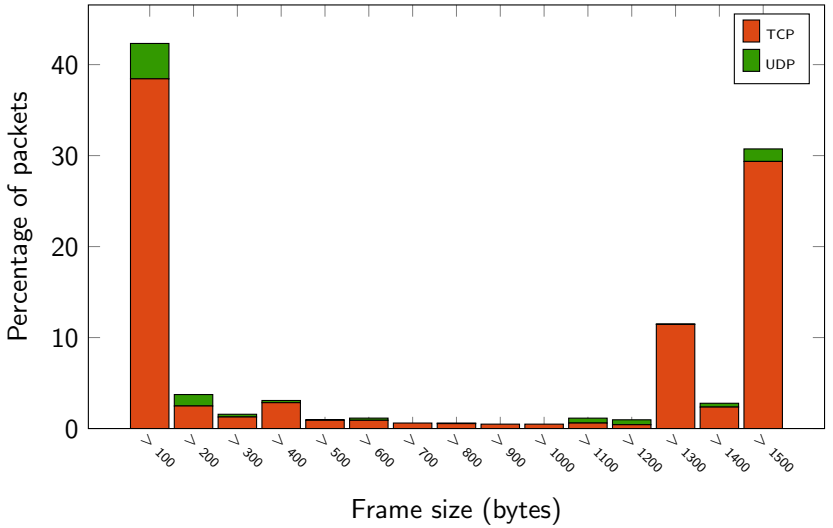
Reality Check: Typical Internet Traffic



Bimodal distribution

- 44% between 40 and 100 bytes
- 37% between 1400 and 1500 bytes

Reality Check: Typical Internet Traffic



Bimodal distribution

- 44% between 40 and 100 bytes
- 37% between 1400 and 1500 bytes

Implications

- Very long messages **unrealistic**
- Many **smaller** messages, highly **clustered**

Filling the Pipeline: Multiple Messages

Observations

- Common: More than one message available at the same time
- Blocks from different messages are independent!

Filling the Pipeline: Multiple Messages

Observations

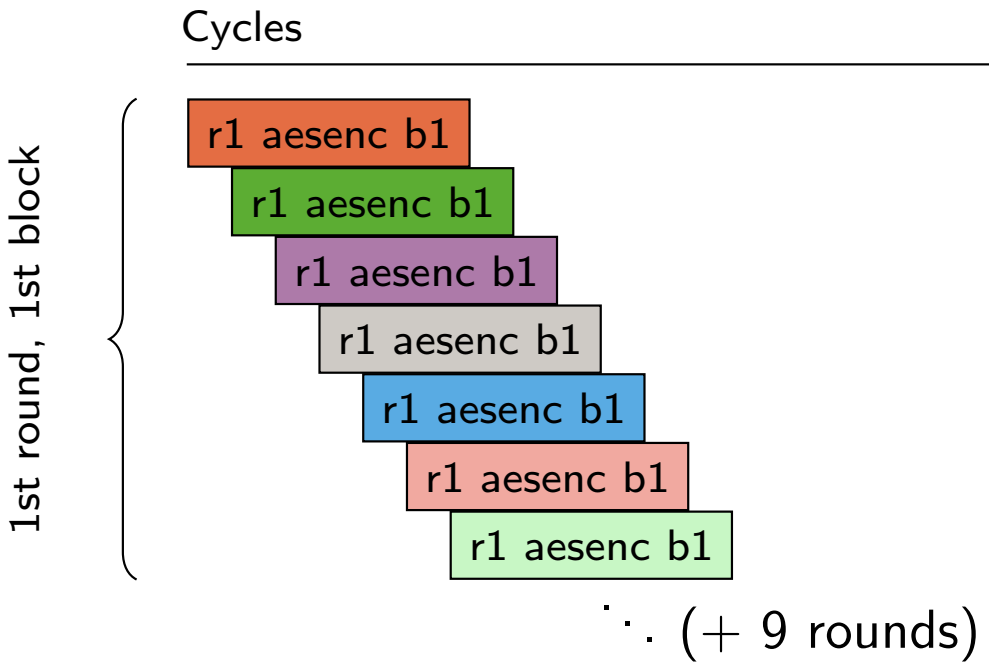
- Common: More than one message available at the same time
- Blocks from different messages are independent!

Speed up serial modes

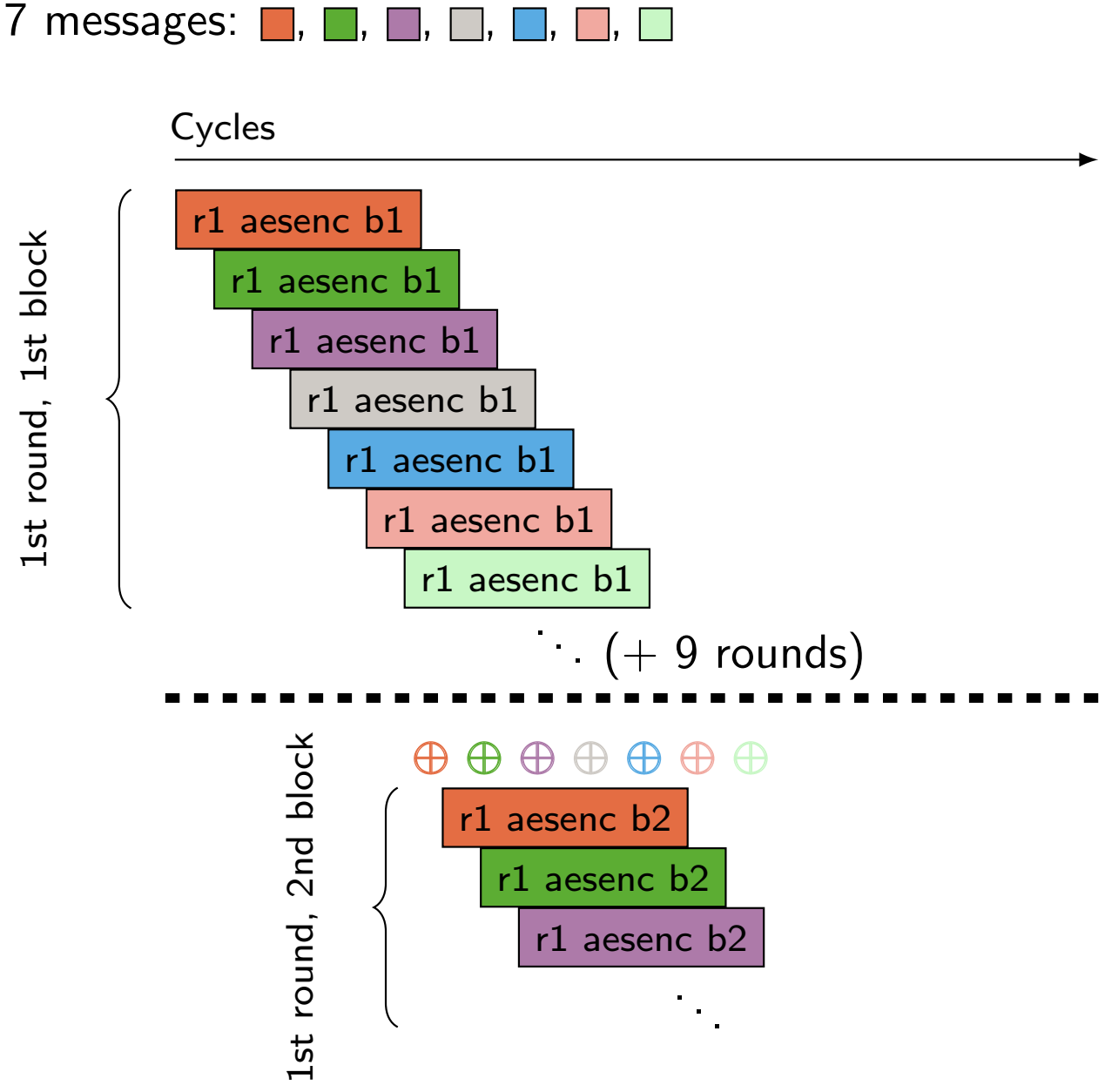
- Process **independent messages** in a **pipeline**!

Filling the Pipeline: CBC with Multiple Messages

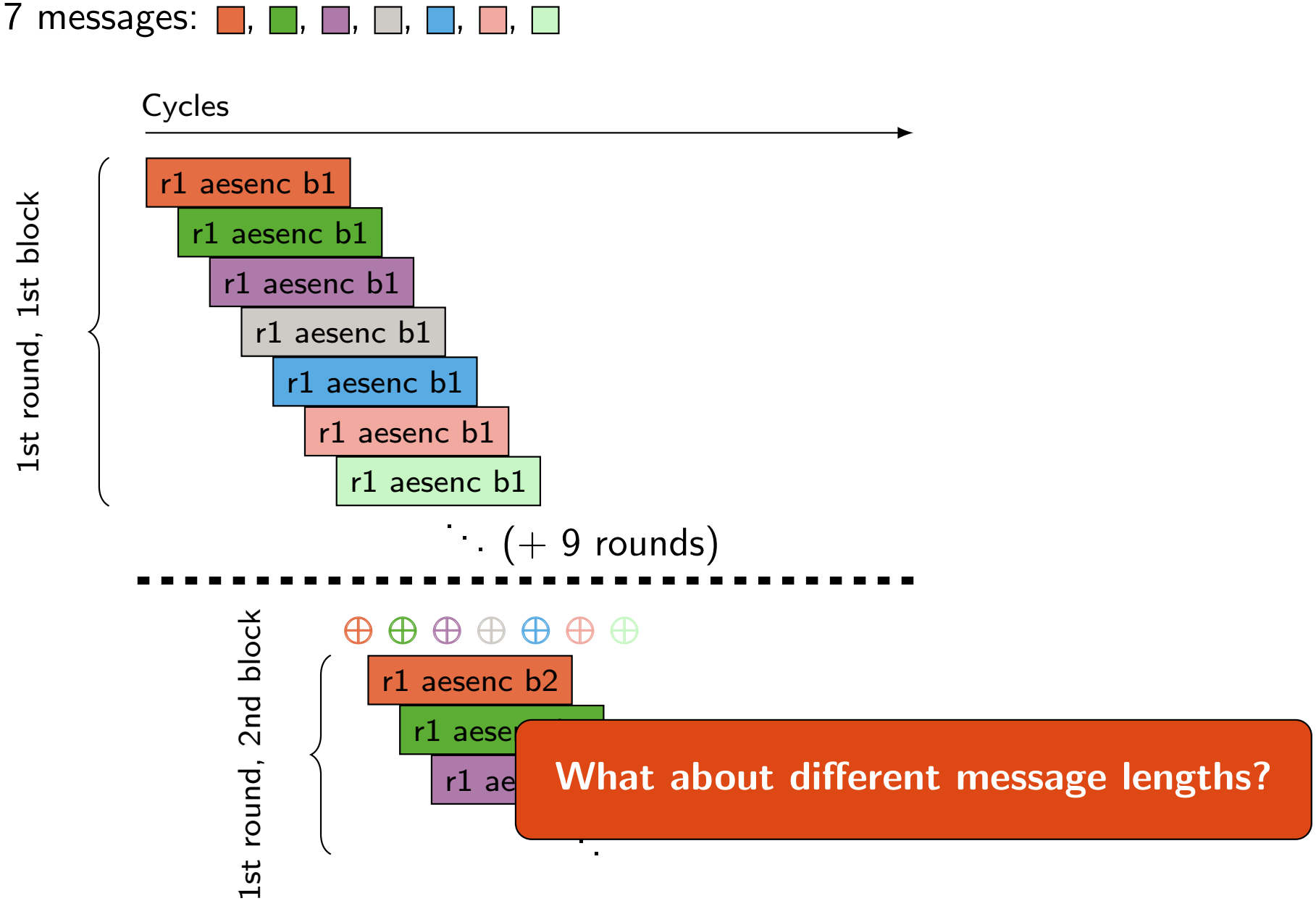
7 messages: ■, ■, ■, ■, ■, ■, ■



Filling the Pipeline: CBC with Multiple Messages



Filling the Pipeline: CBC with Multiple Messages



Combing: Handling Different Message Lengths

Scenario

We are given messages $m_1, m_2, \ldots$ of lengths $\ell_1, \ell_2, \ldots$.

Combing: Handling Different Message Lengths

Scenario

We are given messages $m_1, m_2, \ldots$ of lengths $\ell_1, \ell_2, \ldots$.

Problem

Scheduling decisions $\Rightarrow$ pipeline stall

Combing: Handling Different Message Lengths

Scenario

We are given messages $m_1, m_2, \ldots$ of lengths $\ell_1, \ell_2, \ldots$.

Problem

Scheduling decisions $\Rightarrow$ pipeline stall

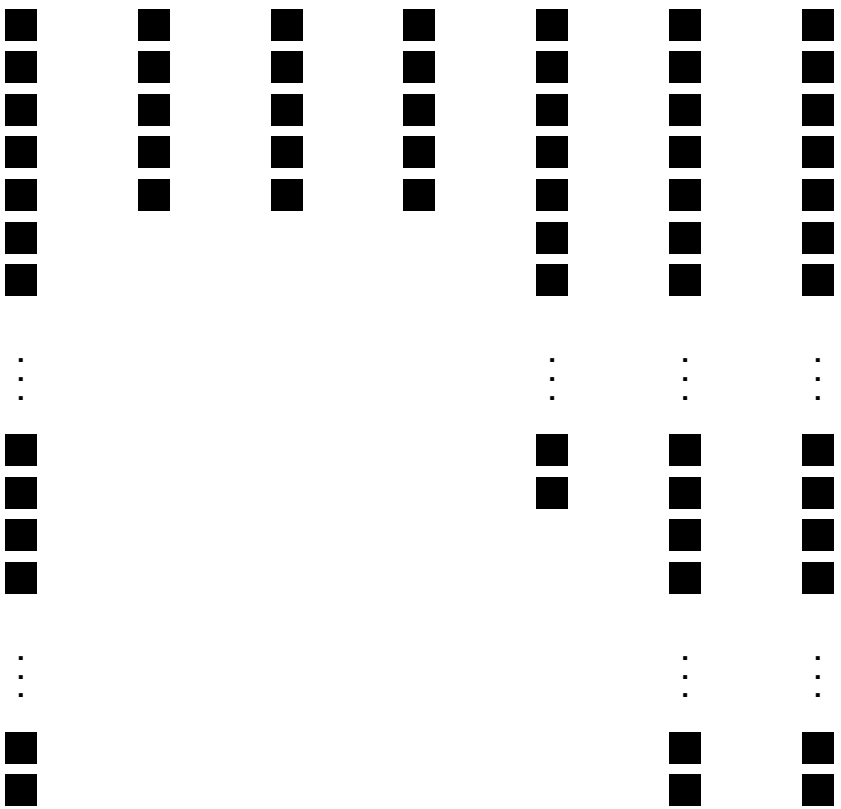
Proposed solution: Comb Scheduling

- **Pre-computation** of scheduling decisions
- **Look-ahead** in windows of width P
- Greedy processing of blocks
- Works for **any** distribution of message lengths

Comb Scheduling: Pre-computation Example

- $P = 7$ messages of lengths $(1504, 80, 80, 80, 1360, 1504, 1504)$ bytes

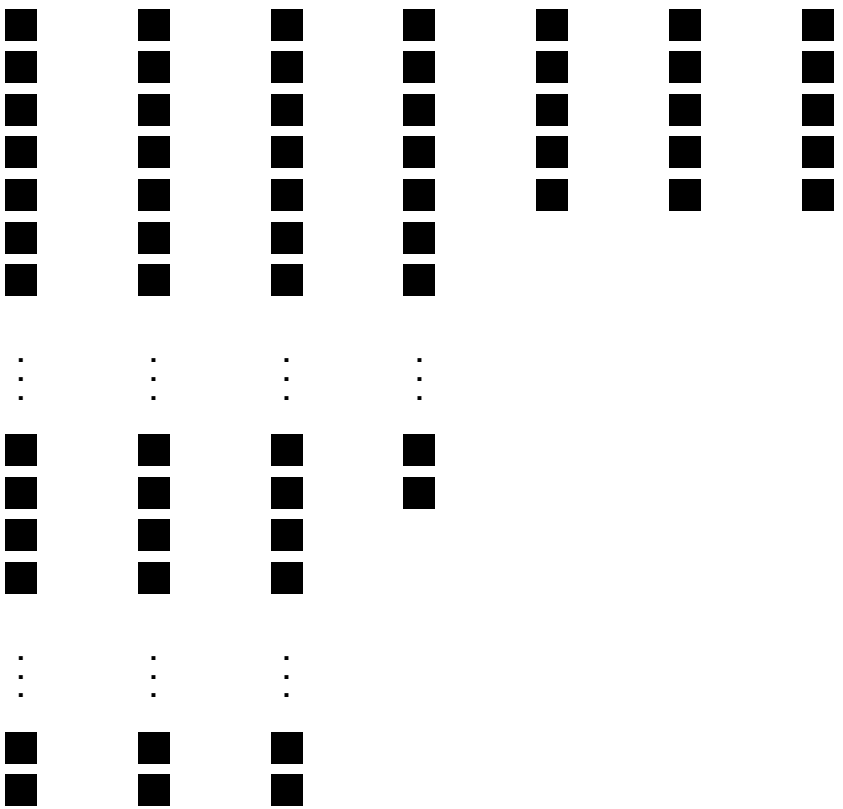
Message	m_1	m_2	m_3	m_4	m_5	m_6	m_7	Sub-windows
Length	94	5	5	5	85	94	94	$(\mathcal{P}[w], \mathcal{B}[w])$



Comb Scheduling: Pre-computation Example

- $P = 7$ messages of lengths $(1504, 80, 80, 80, 1360, 1504, 1504)$ bytes

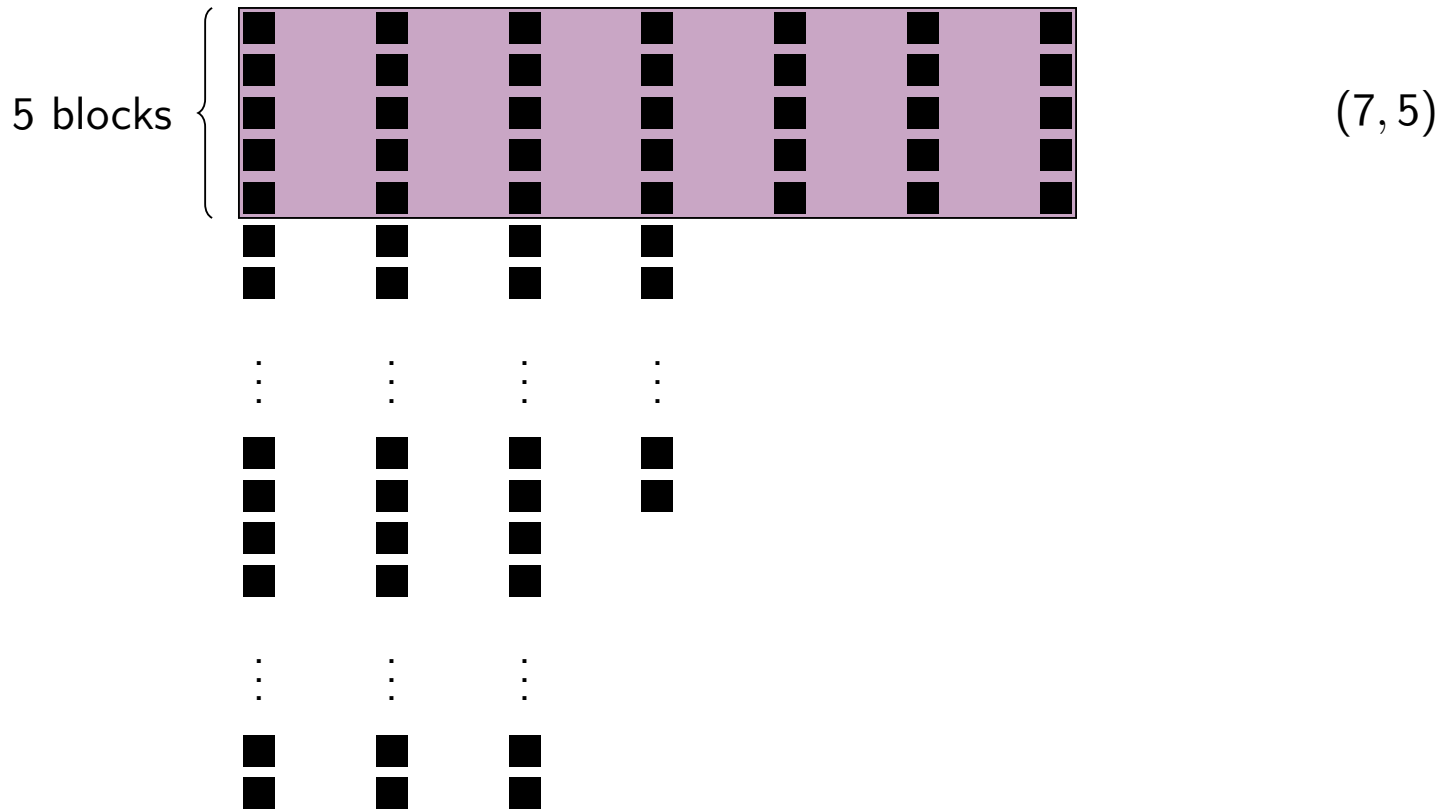
Message	m_1	m_6	m_7	m_5	m_2	m_3	m_4	Sub-windows
Length	94	94	94	85	5	5	5	$(\mathcal{P}[w], \mathcal{B}[w])$



Comb Scheduling: Pre-computation Example

- $P = 7$ messages of lengths $(1504, 80, 80, 80, 1360, 1504, 1504)$ bytes

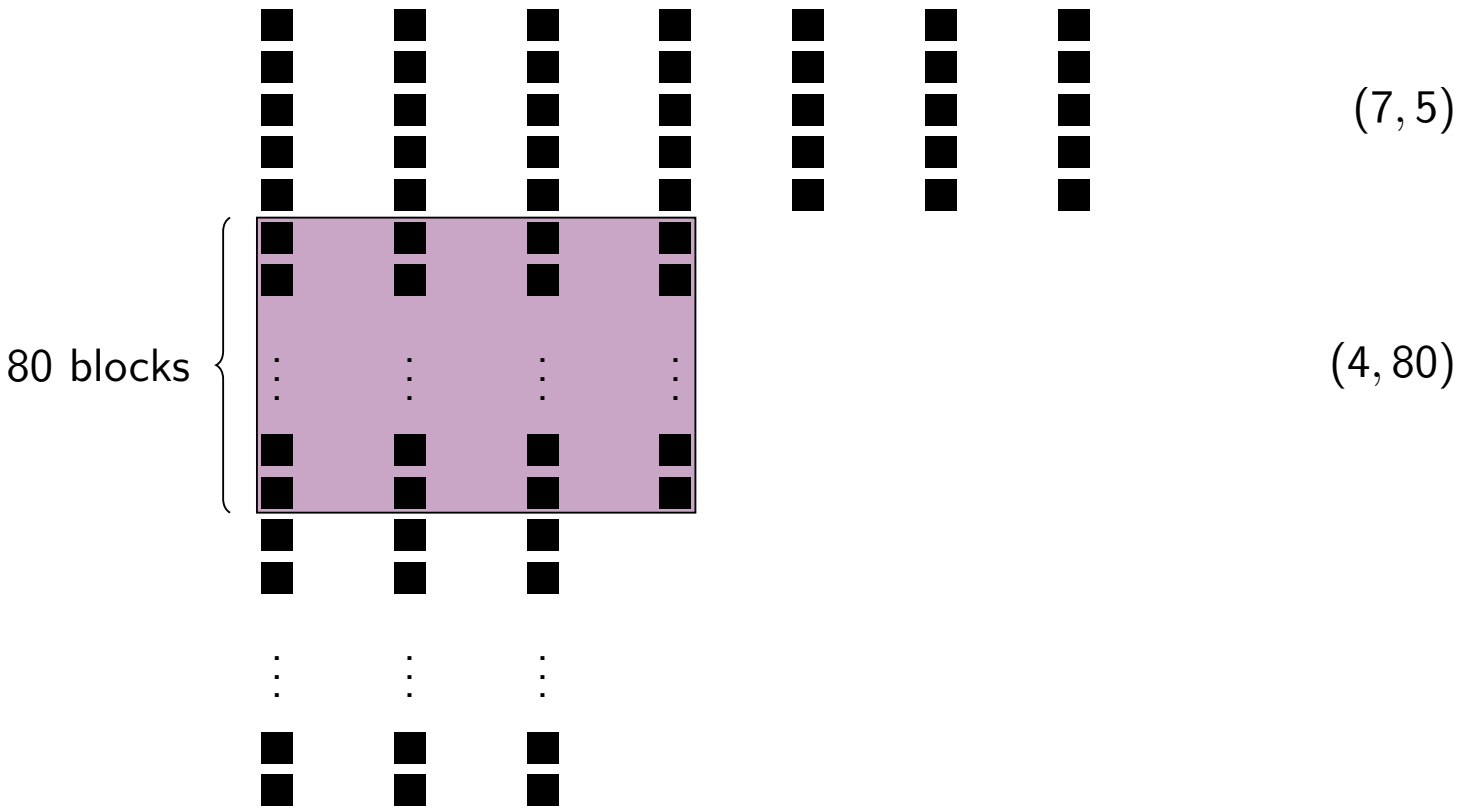
Message	m_1	m_6	m_7	m_5	m_2	m_3	m_4	Sub-windows
Length	94	94	94	85	5	5	5	$(\mathcal{P}[w], \mathcal{B}[w])$



Comb Scheduling: Pre-computation Example

- $P = 7$ messages of lengths $(1504, 80, 80, 80, 1360, 1504, 1504)$ bytes

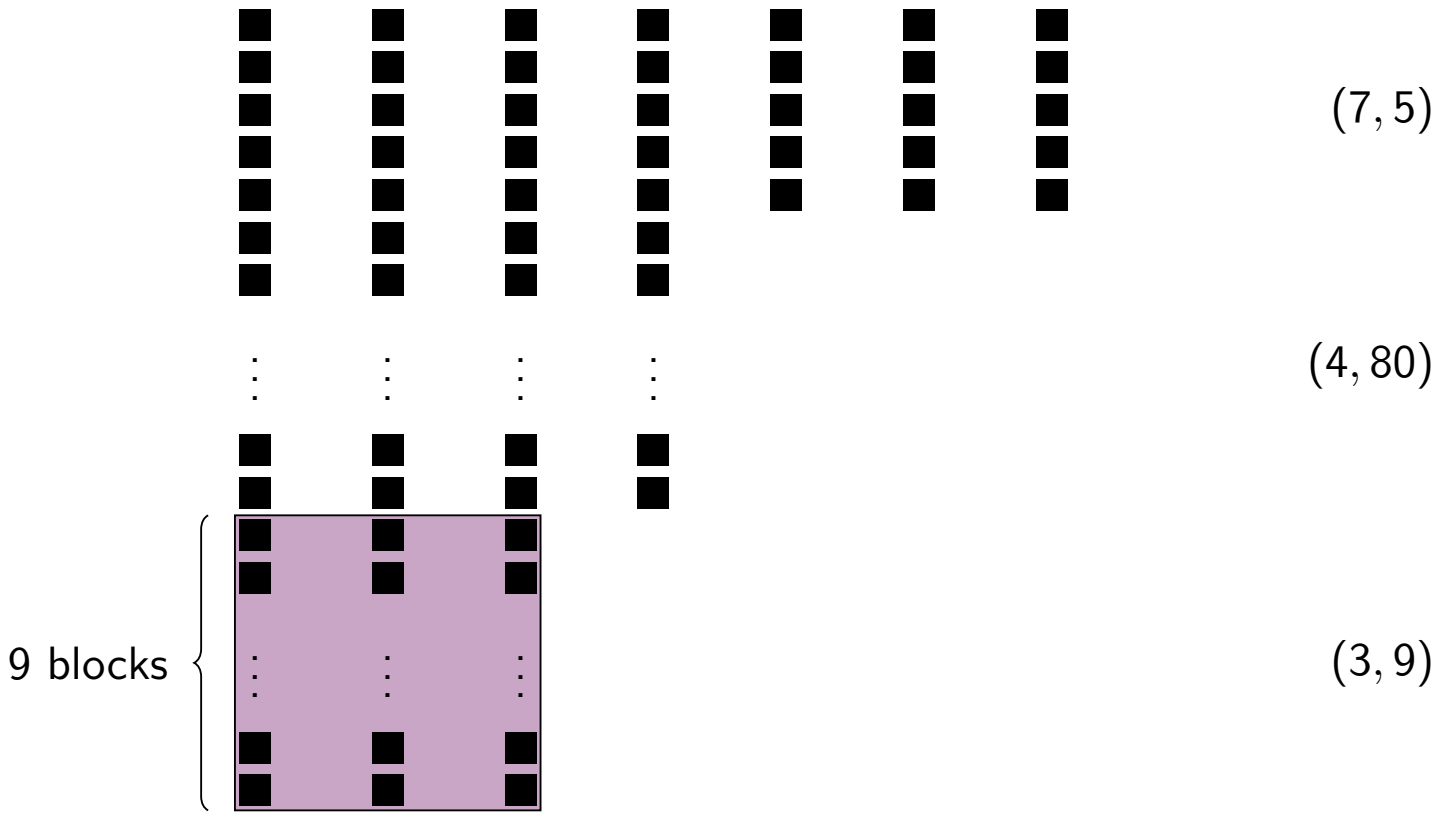
Message	m_1	m_6	m_7	m_5	m_2	m_3	m_4	Sub-windows
Length	94	94	94	85	5	5	5	$(\mathcal{P}[w], \mathcal{B}[w])$



Comb Scheduling: Pre-computation Example

- $P = 7$ messages of lengths $(1504, 80, 80, 80, 1360, 1504, 1504)$ bytes

Message	m_1	m_6	m_7	m_5	m_2	m_3	m_4	Sub-windows
Length	94	94	94	85	5	5	5	$(\mathcal{P}[w], \mathcal{B}[w])$



Showcase: Comb Scheduling for CBC mode

	Serial	Window size P						
		2	3	4	5	6	7	8
2KB msg.	4.38	2.19	1.47	1.11	0.91	0.76	0.66	0.65
Speed-up	× 1.00	×2.00	×2.98	×3.95	×4.81	×5.76	× 6.64	×6.74
Msg. mix	4.38	2.42	1.73	1.37	1.08	0.98	0.87	0.85
Speed-up	× 1.00	×1.81	×2.53	×3.20	×4.06	×4.47	× 5.03	×5.15

Showcase: Comb Scheduling for CBC mode

	Serial	Window size P						
		2	3	4	5	6	7	8
2KB msg.	4.38	2.19	1.47	1.11	0.91	0.76	0.66	0.65
Speed-up	× 1.00	×2.00	×2.98	×3.95	×4.81	×5.76	× 6.64	×6.74
Msg. mix	4.38	2.42	1.73	1.37	1.08	0.98	0.87	0.85
Speed-up	× 1.00	×1.81	×2.53	×3.20	×4.06	×4.47	× 5.03	×5.15

- **Fixed** lengths (2KB): 5% latency increase for ×6.6 speed-up

Showcase: Comb Scheduling for CBC mode

	Serial	Window size P						
		2	3	4	5	6	7	8
2KB msg.	4.38	2.19	1.47	1.11	0.91	0.76	0.66	0.65
Speed-up	× 1.00	×2.00	×2.98	×3.95	×4.81	×5.76	× 6.64	×6.74
Msg. mix	4.38	2.42	1.73	1.37	1.08	0.98	0.87	0.85
Speed-up	× 1.00	×1.81	×2.53	×3.20	×4.06	×4.47	× 5.03	×5.15

- **Fixed** lengths (2KB): 5% latency increase for ×6.6 speed-up
- **Realistic** lengths: 39% latency increase for ×5 speed-up

Showcase: Comb Scheduling for CBC mode

	Serial	Window size P						
		2	3	4	5	6	7	8
2KB msg.	4.38	2.19	1.47	1.11	0.91	0.76	0.66	0.65
Speed-up	× 1.00	×2.00	×2.98	×3.95	×4.81	×5.76	× 6.64	×6.74
Msg. mix	4.38	2.42	1.73	1.37	1.08	0.98	0.87	0.85
Speed-up	× 1.00	×1.81	×2.53	×3.20	×4.06	×4.47	× 5.03	×5.15

- **Fixed** lengths (2KB): 5% latency increase for ×6.6 speed-up
- **Realistic** lengths: 39% latency increase for ×5 speed-up
  - **Throughput**: Within **10 – 30%** of optimal speed-up

Pipelined Authenticated Encryption Modes

Mode	Sequential	**Comb Scheduling**	Speed-up
CCM	5.22	**1.64**	×3.18
GCM	1.63	—	—
OCB3	1.51	—	—
OTR	1.91	—	—
COBRA	3.56	—	—
CLOC	4.47	**1.45**	×3.08
JAMBU	9.12	**2.05**	×4.45
SILC	4.53	**1.49**	×3.04
McOE-G	7.41	**1.79**	×4.14
COPA	2.68	—	—
POET	5.85	**2.14**	×2.73
Julius	3.73	—	—

Conclusions and Outlook

Conclusions

- Realistic Internet traffic means many small(er) messages
- **Comb Scheduler**: New efficient algorithm for pipelining otherwise serial modes
- Serial modes suddenly competitive again

Conclusions and Outlook

Conclusions

- Realistic Internet traffic means many small(er) messages
- **Comb Scheduler**: New efficient algorithm for pipelining otherwise serial modes
- Serial modes suddenly competitive again

Future work

- Implement Comb also for parallelizable AE modes (initialization/finalization)
- Longer windows
- Dynamic window adaptation

The End

Thank you!

Questions?