The Support of Design Patterns for Streaming RPC on Embedded Multicore Processors

Brian Kun-Yuan Hsieh
Yen-Chih Liu
Chi-Hua Lai
Jenq Kuen Lee

Department of Computer Science
National Tsing-Hua University
Hsinchu, Taiwan

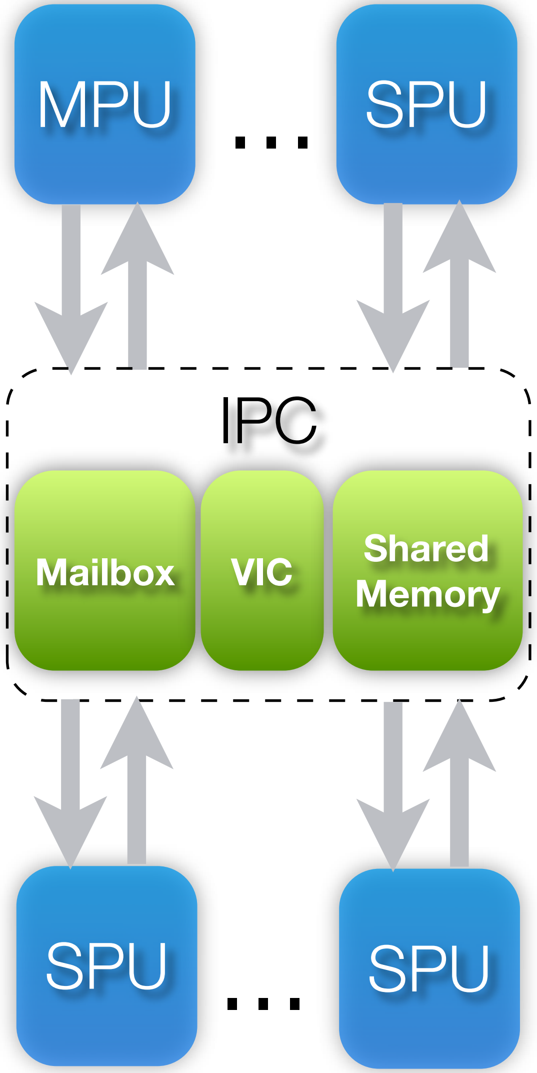
2008 IEEE WORKSHOP ON SIGNAL PROCESSING SYSTEMS OCTOBER 8-10, 2008 WASHINGTON, D.C. METRO AREA, U.S.A.

Outline

- Background & motivation
- Streaming RPC framework
- Software design patterns
- Experimental results
- Summary

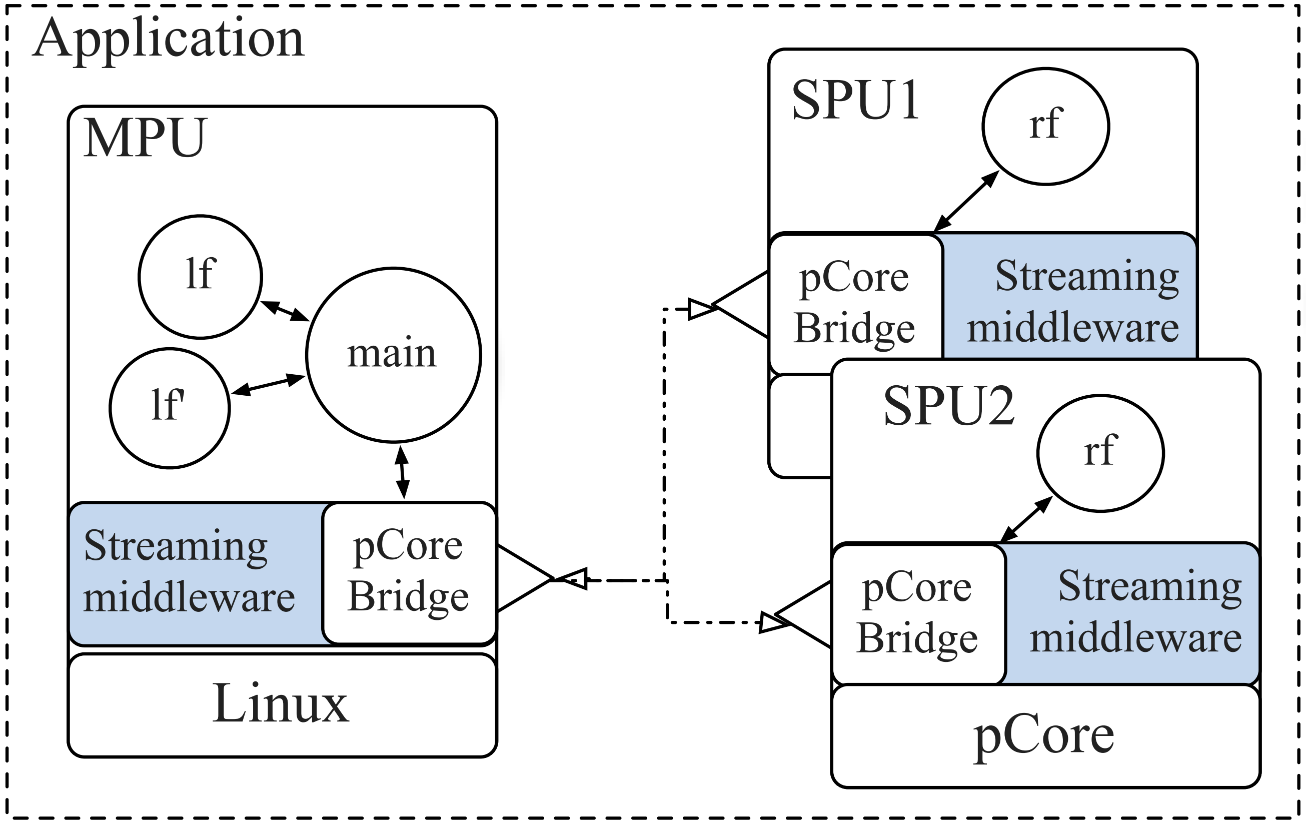
Challenges in Programming MPSoC

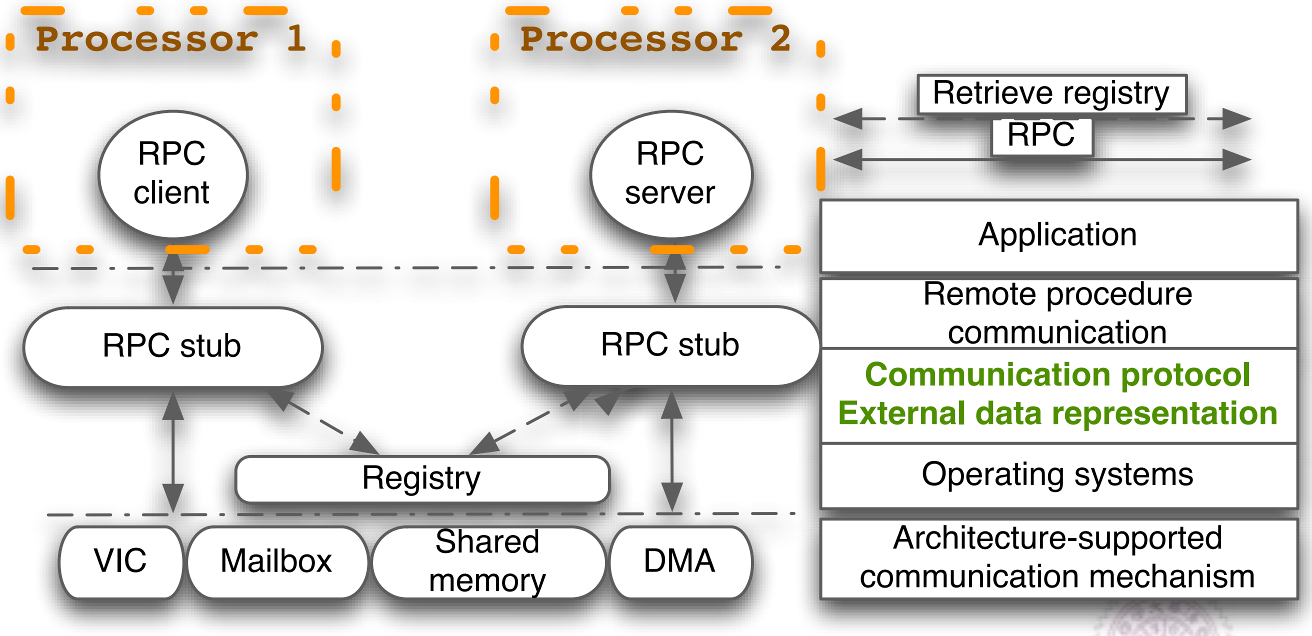
- Multicore processors being widely used in the handheld multimedia devices
- Challenges in writing program raise issues in providing programming model
  - Multiple ISA
  - Various inter-processor communication(IPC)
  - Parallel programming
- Moreover, applications are with data streaming in the multimedia application domain
  - Video encoding&decoding, graphic rendering...
- One important issue is to provide streaming functionality!



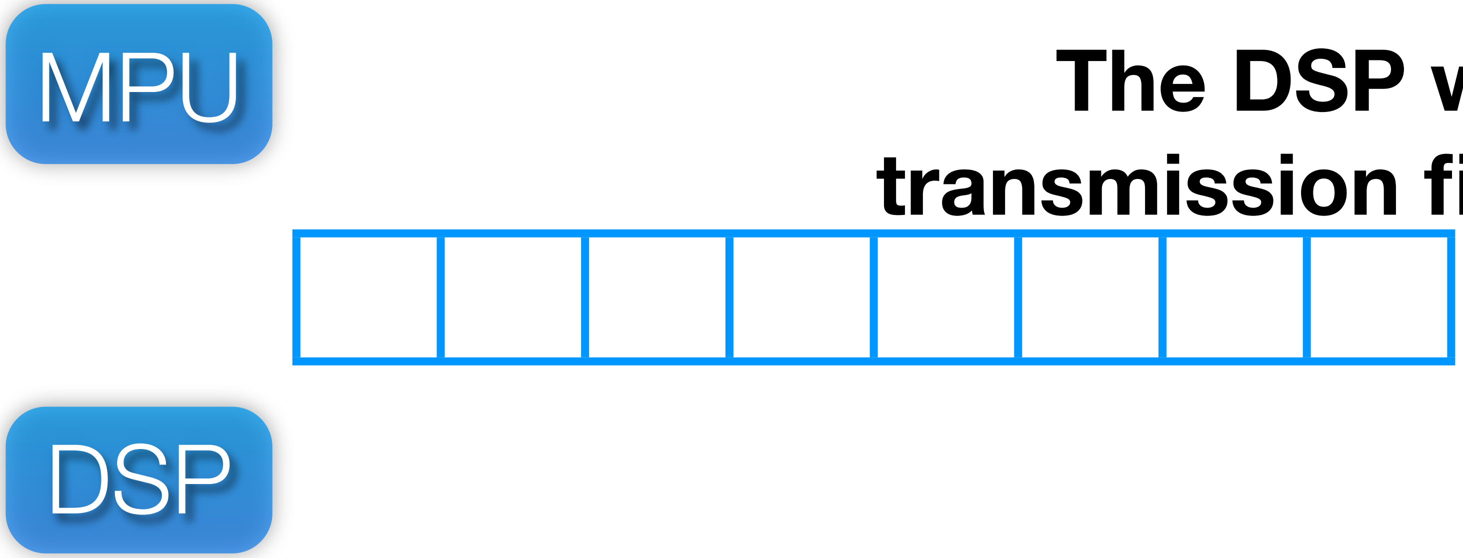
Multicore Programming with RPC

- Model the communication between processors as end-to-end service
- Communicates by invoking commands
- Simple programming model, inefficient in modeling data streaming applications.

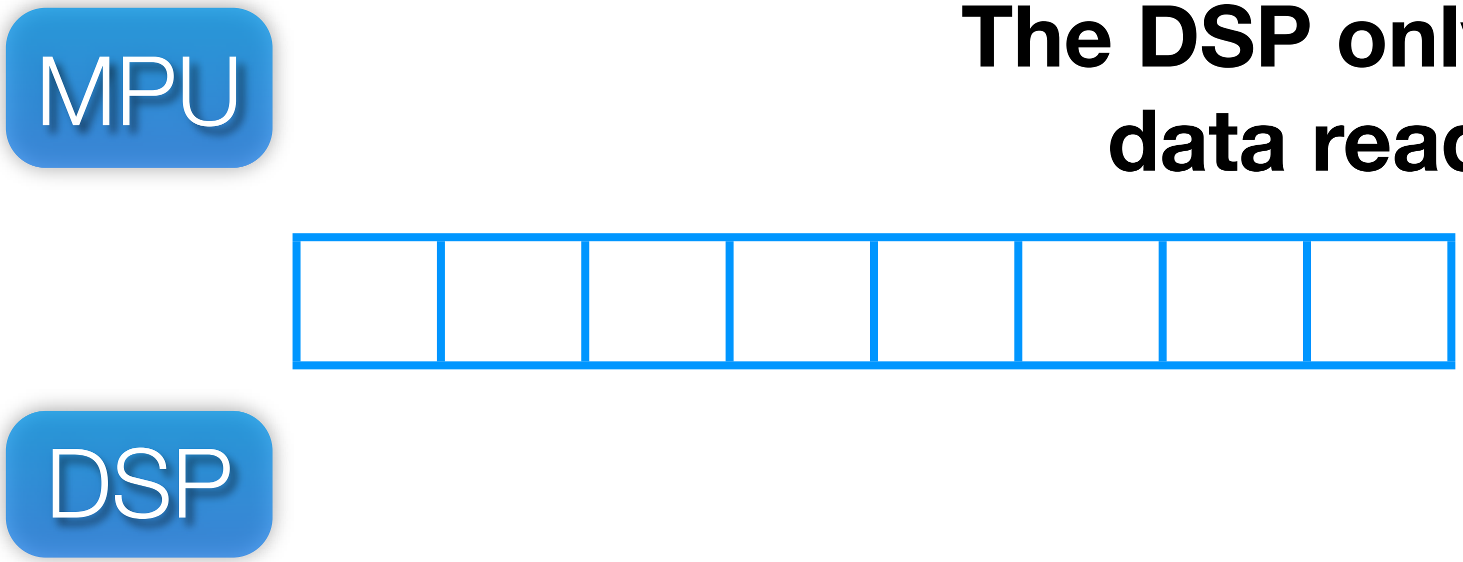




Communication Model of RPC

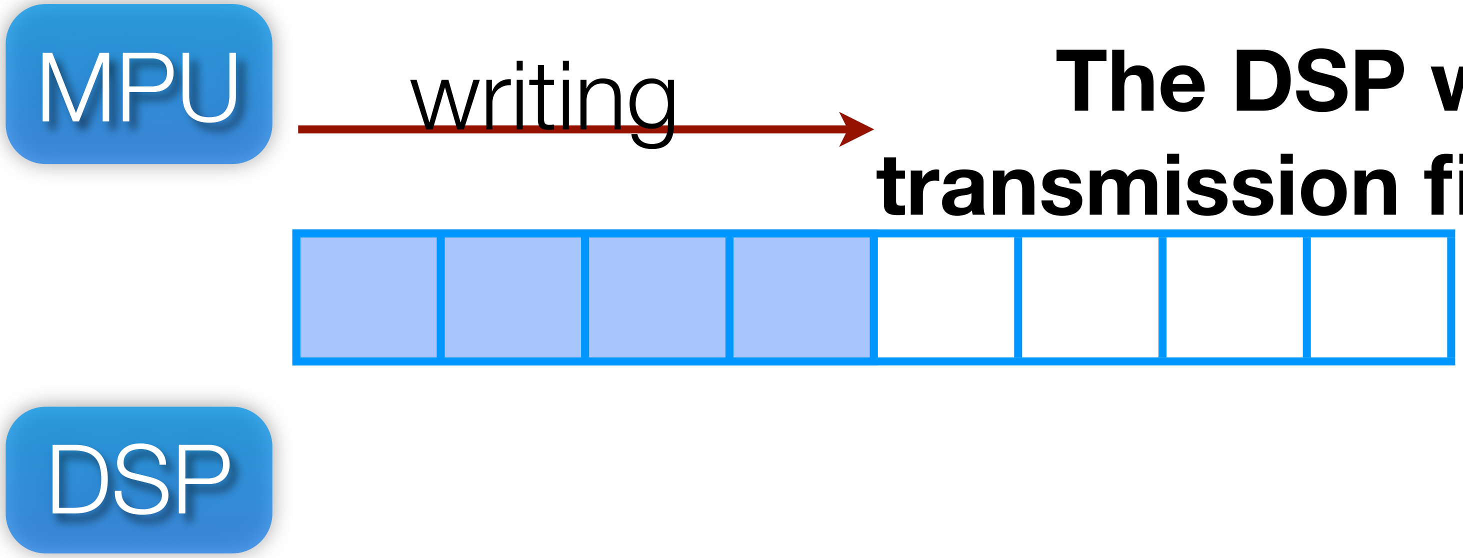


The DSP waits for all the data transmission finish to start processing

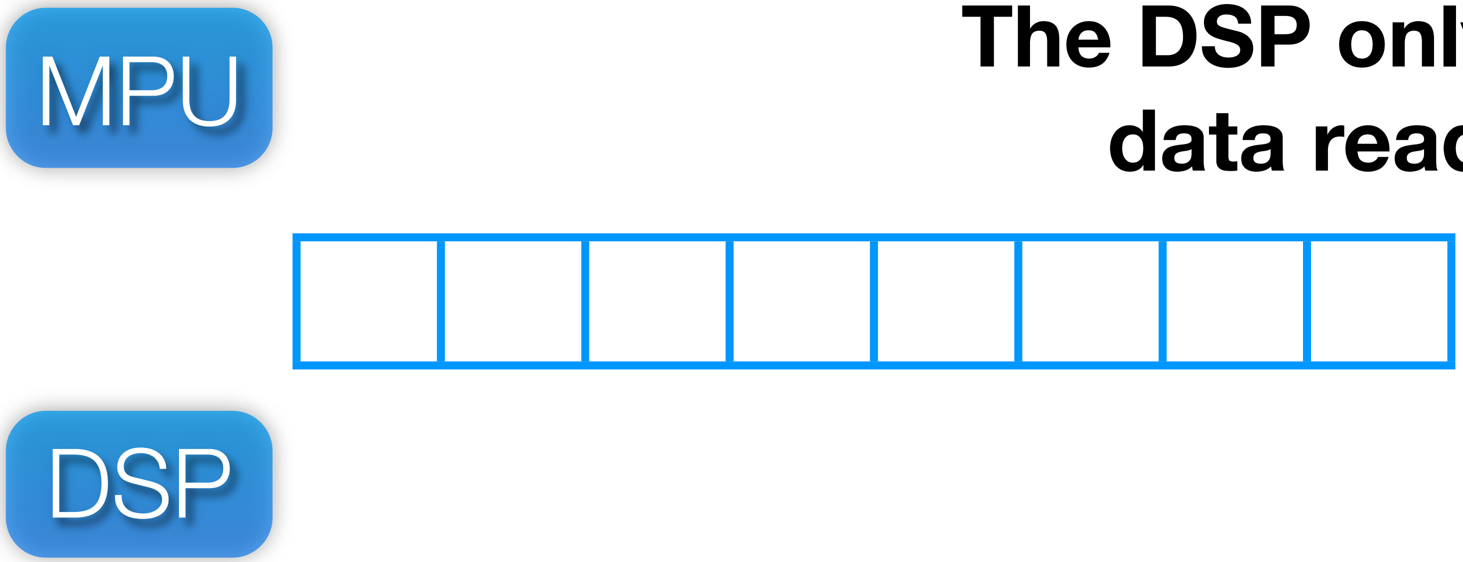


The DSP only waits for the necessary data ready to start processing

Communication Model of RPC

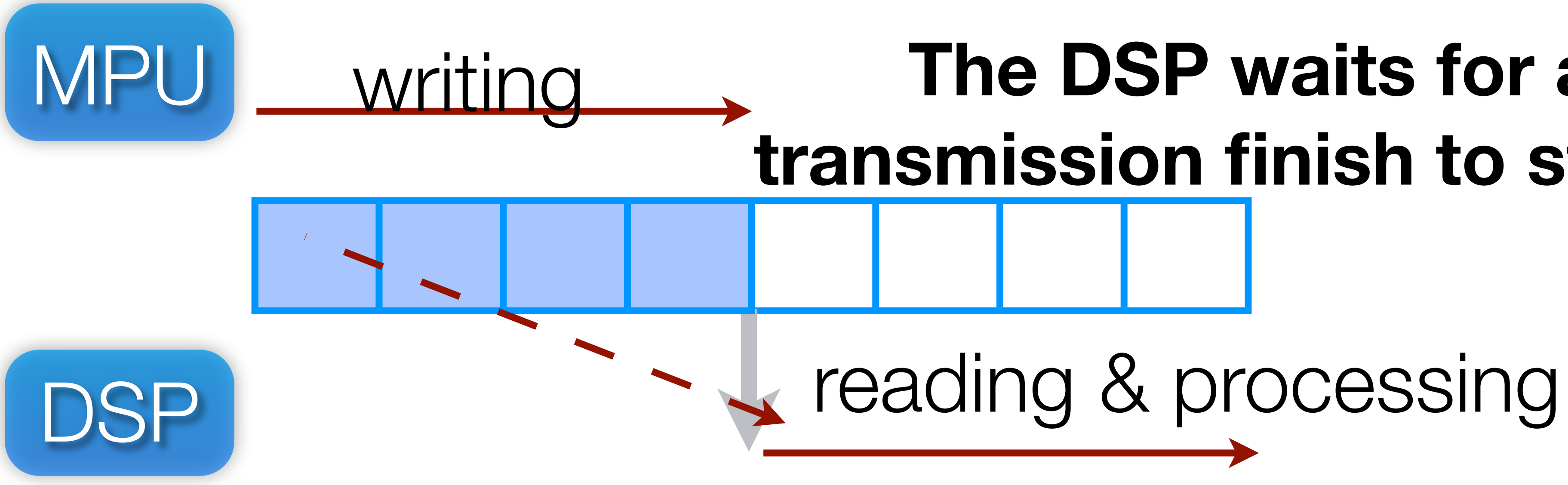


The DSP waits for all the data transmission finish to start processing

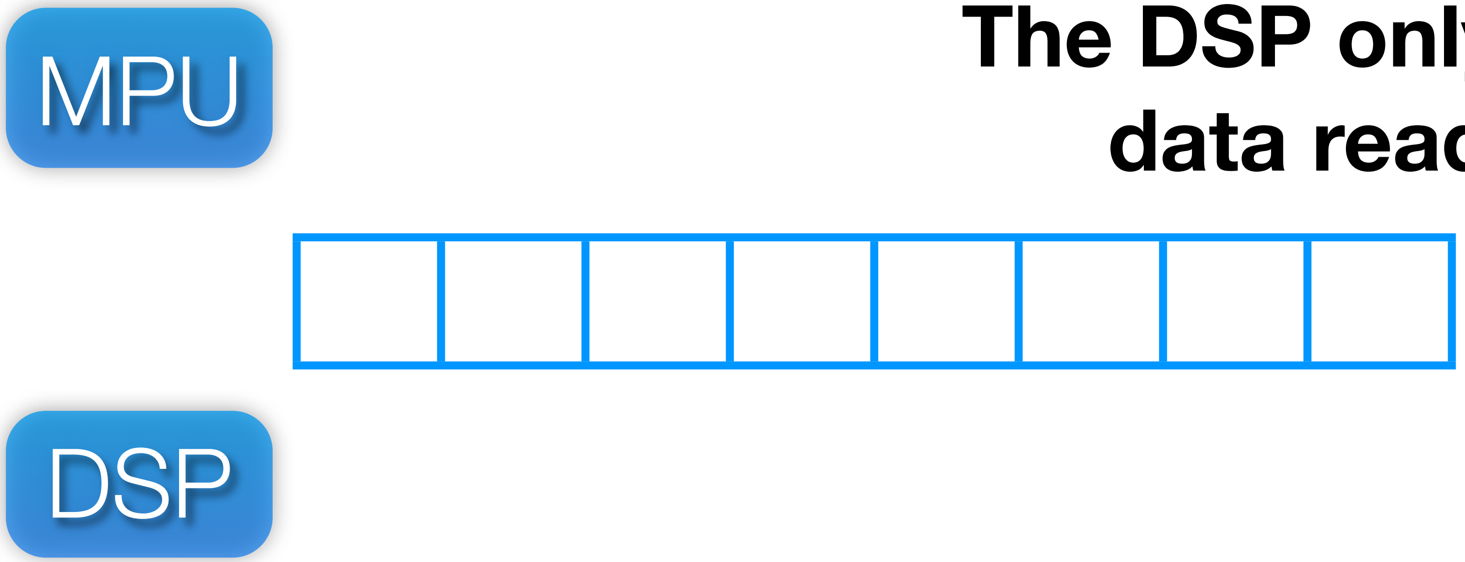


The DSP only waits for the necessary data ready to start processing

Communication Model of RPC

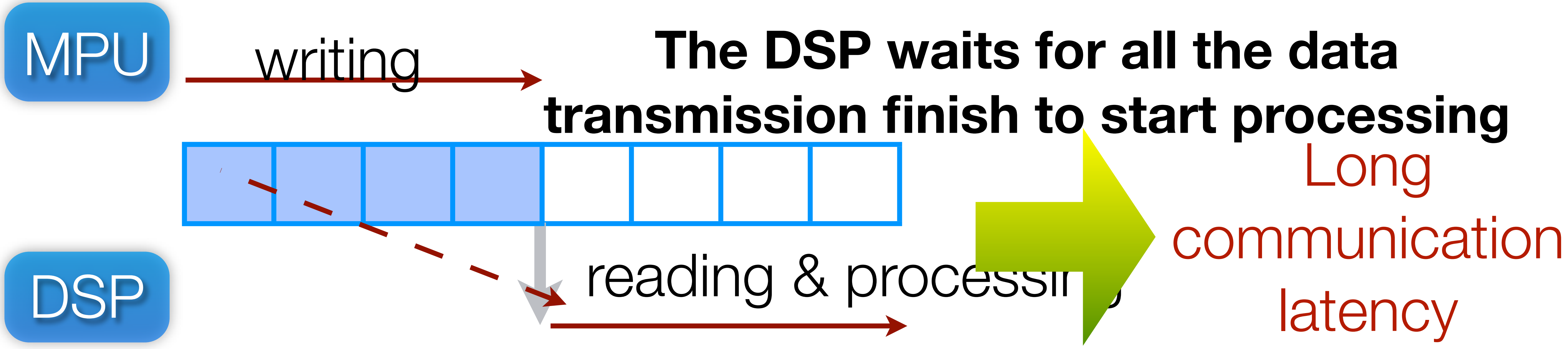


The DSP waits for all the data transmission finish to start processing

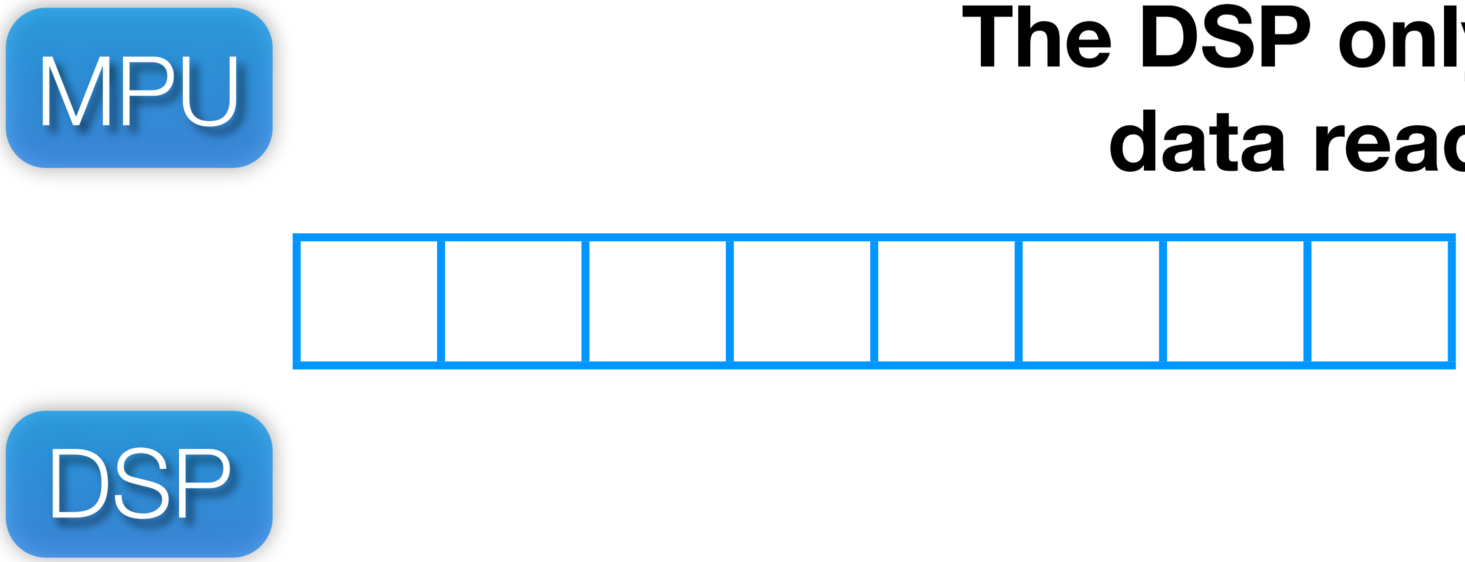


The DSP only waits for the necessary data ready to start processing

Communication Model of RPC

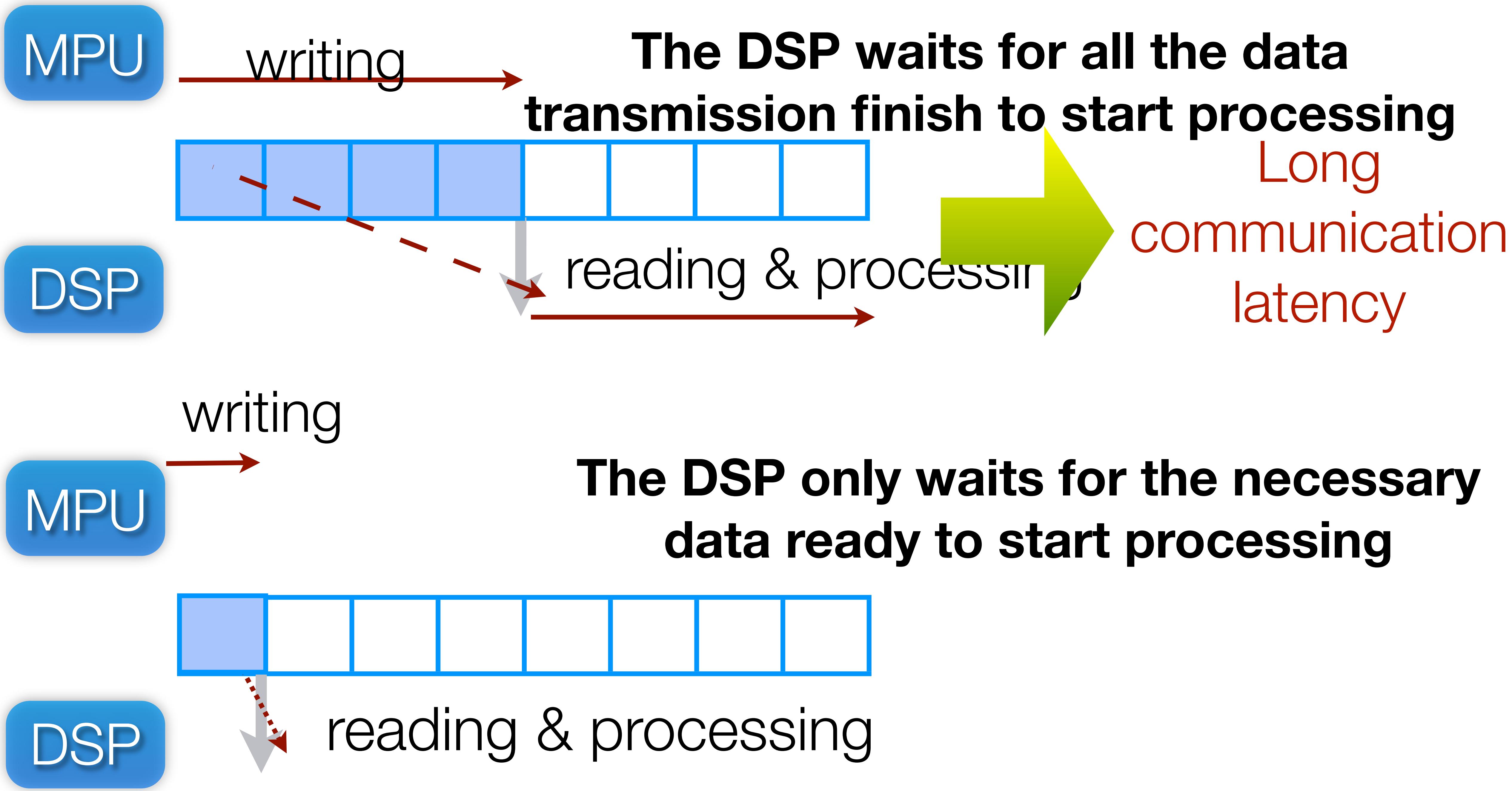


The DSP waits for all the data transmission finish to start processing



The DSP only waits for the necessary data ready to start processing

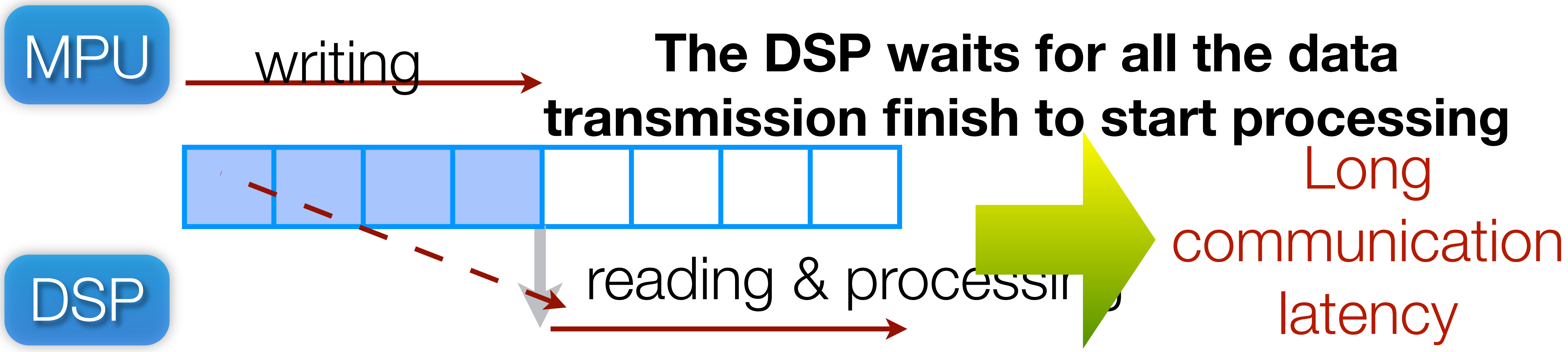
Communication Model of RPC



The DSP waits for all the data transmission finish to start processing

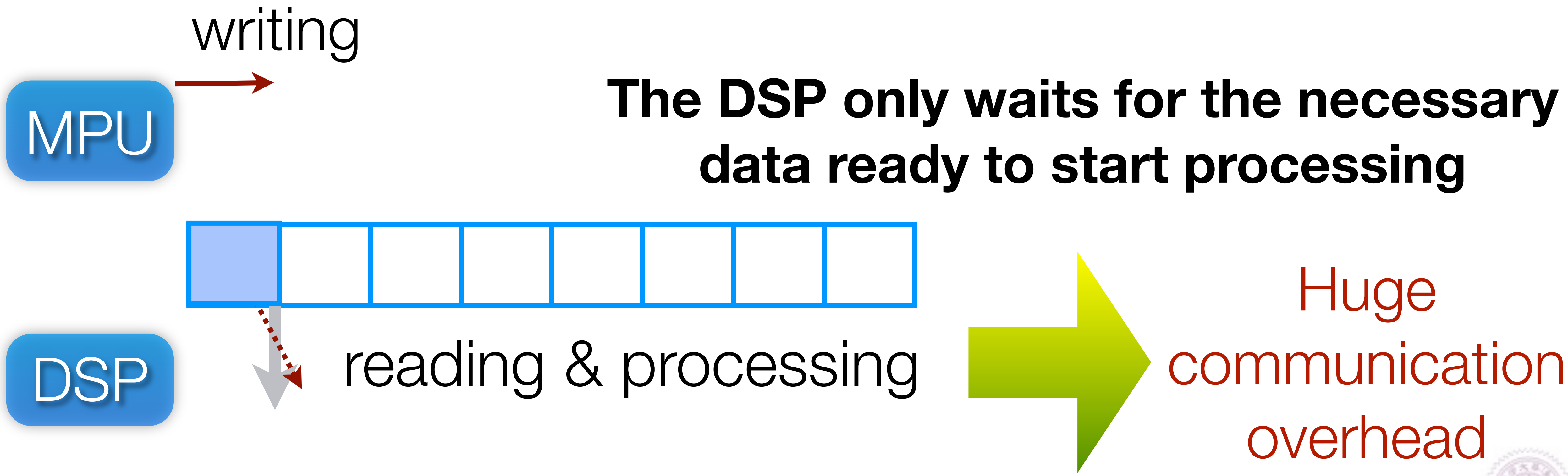
The DSP only waits for the necessary data ready to start processing

Communication Model of RPC



The DSP waits for all the data transmission finish to start processing

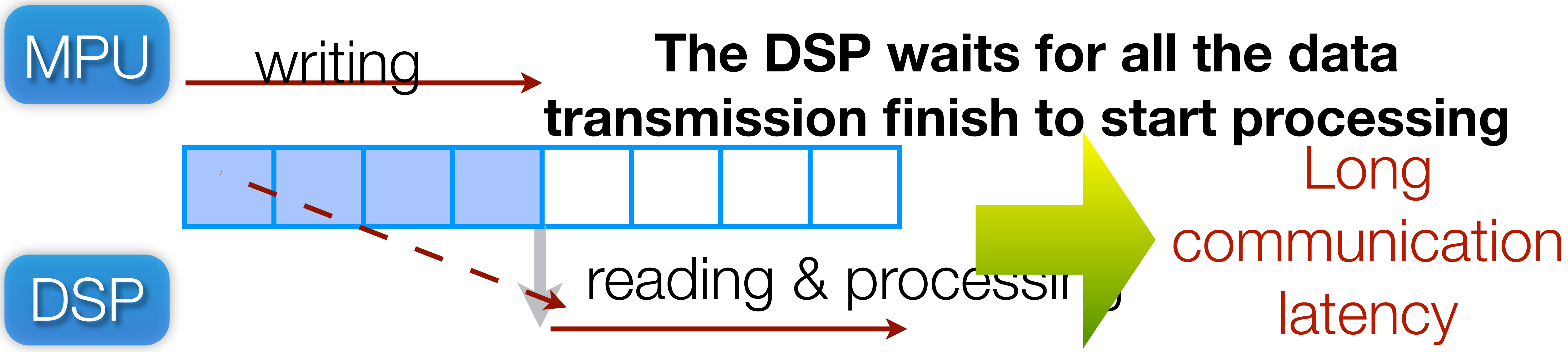
Long communication latency



The DSP only waits for the necessary data ready to start processing

Huge communication overhead

Communication Model of RPC



MPU

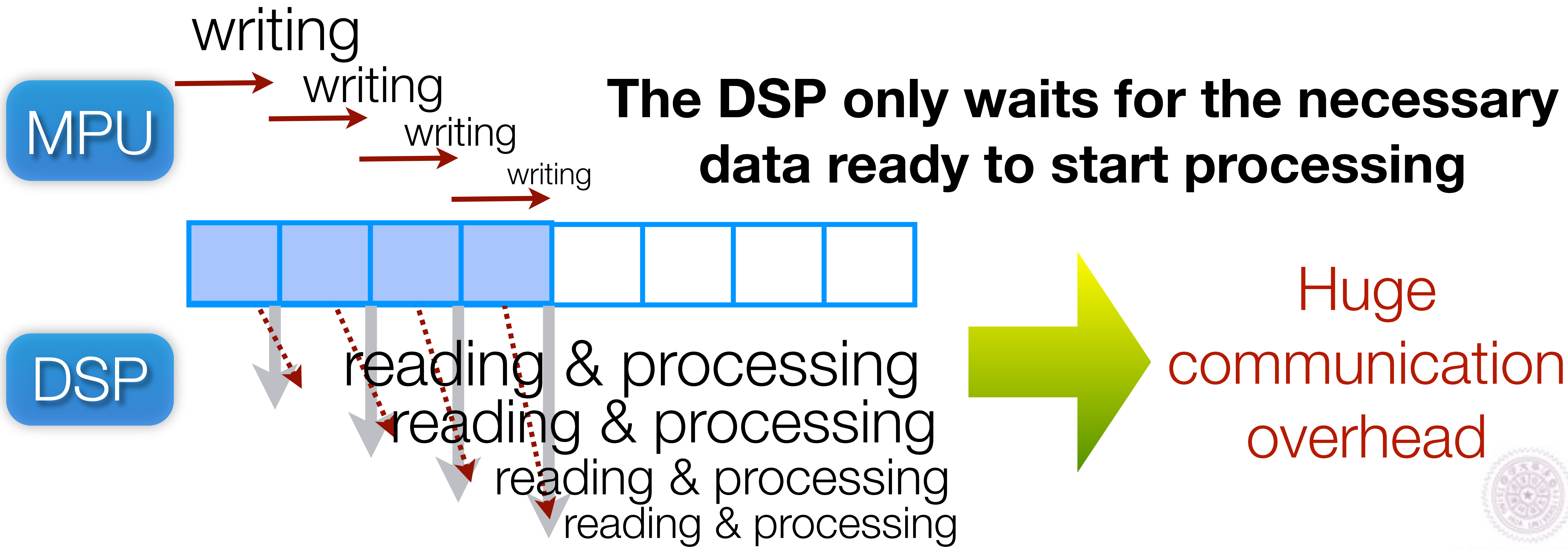
writing

DSP

reading & processing

The DSP waits for all the data transmission finish to start processing

Long communication latency



MPU

writing

writing

writing

writing

DSP

reading & processing

reading & processing

reading & processing

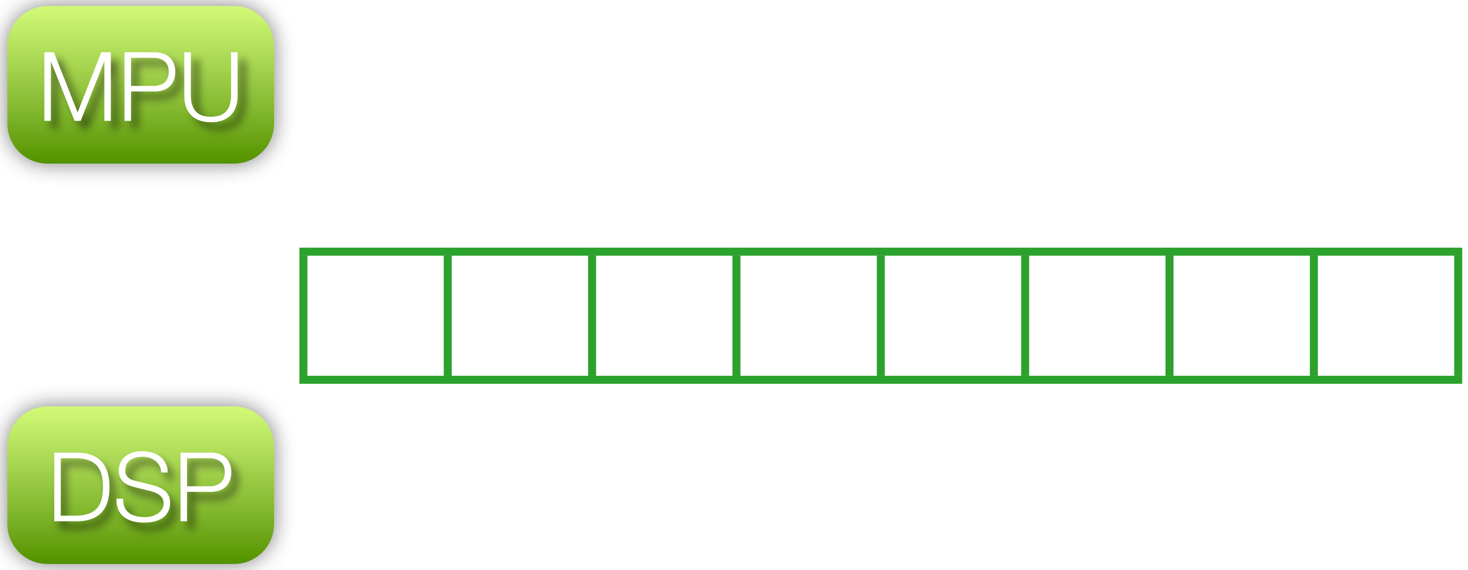
reading & processing

The DSP only waits for the necessary data ready to start processing

Huge communication overhead

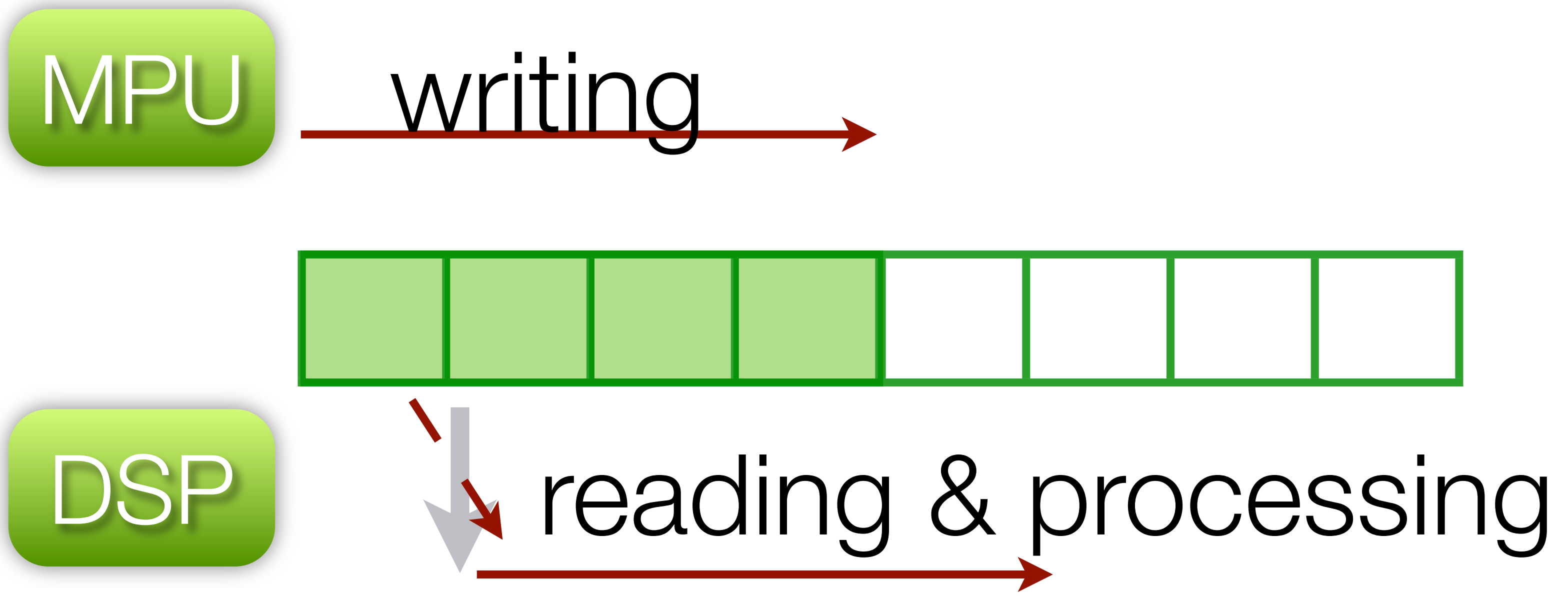
Communication Model of Streaming RPC

- Efficient communication mechanism for streaming applications
- Reducing the handshaking times
- Overlapping communication and computation



Communication Model of Streaming RPC

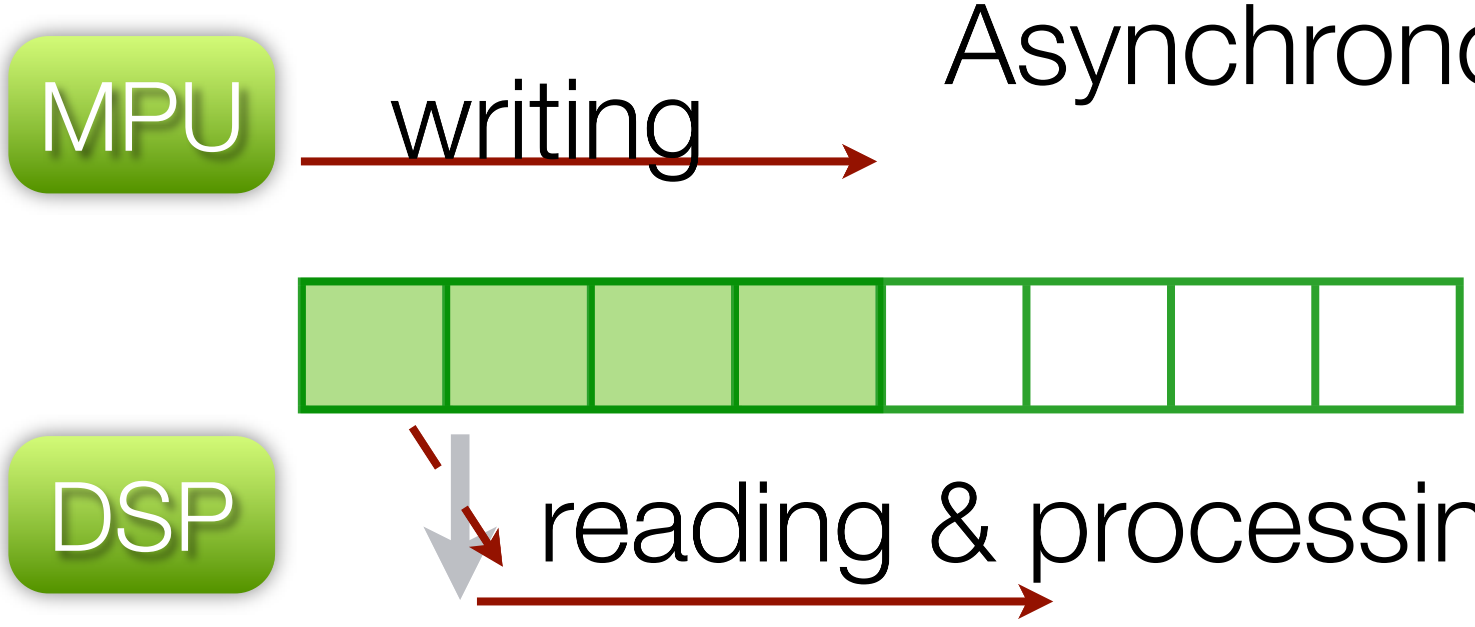
- Efficient communication mechanism for streaming applications
- Reducing the handshaking times
- Overlapping communication and computation



Communication Model of Streaming RPC

- Efficient communication mechanism for streaming applications
- Reducing the handshaking times
- Overlapping communication and computation

Asynchronous invocation



MPU

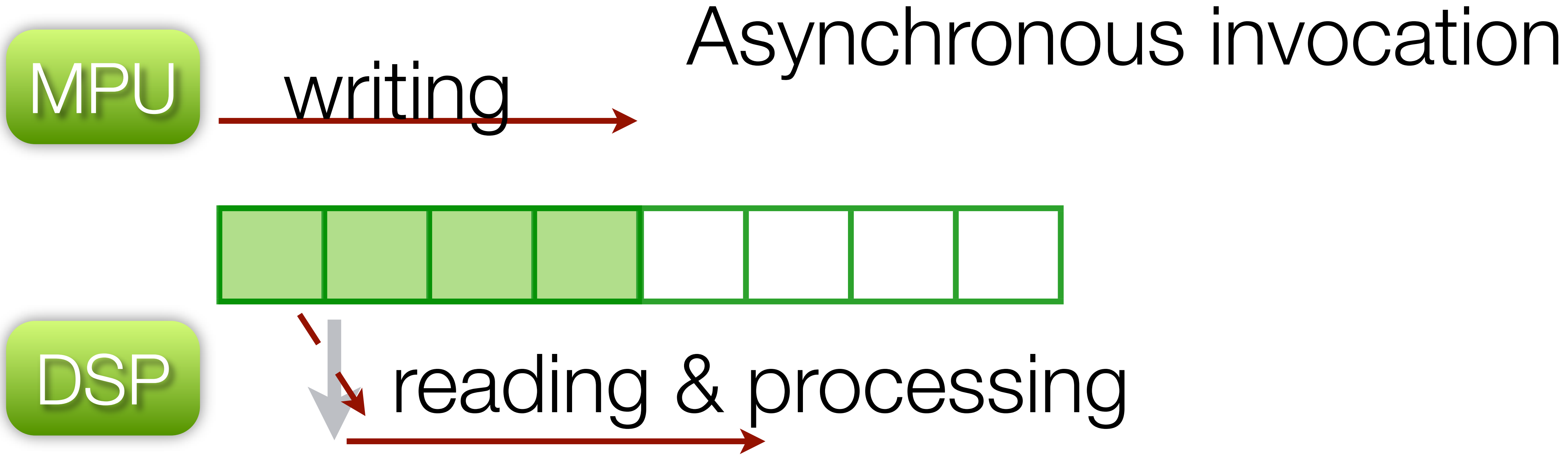
writing

DSP

reading & processing

Communication Model of Streaming RPC

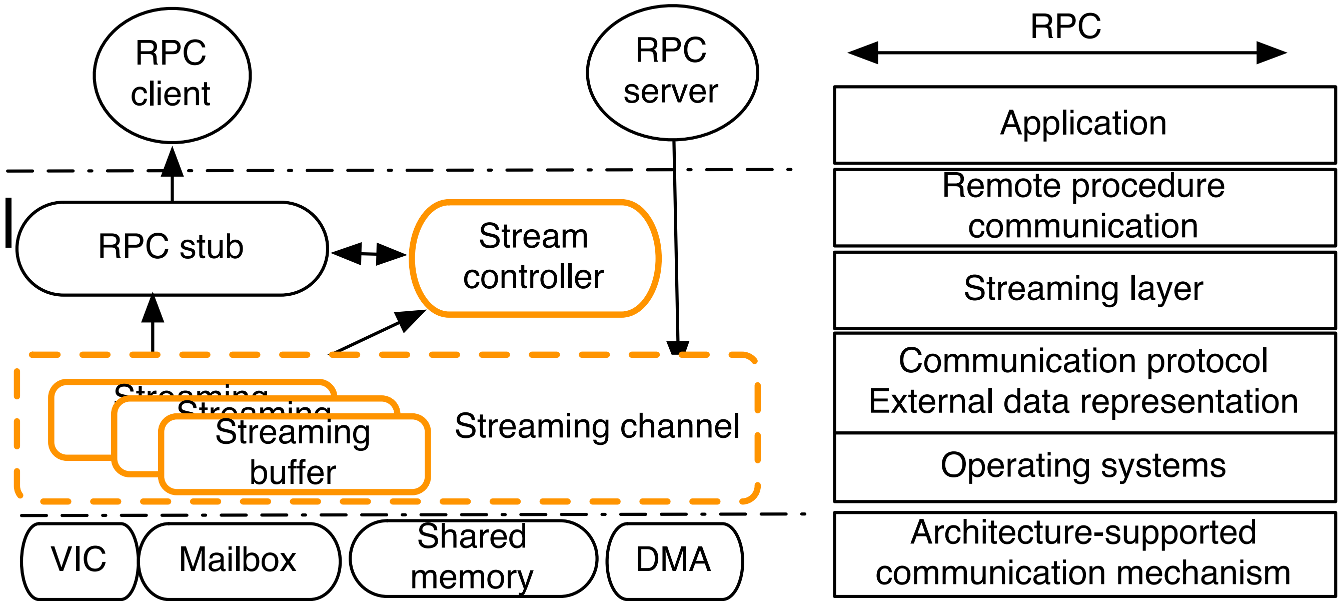
- Efficient communication mechanism for streaming applications
- Reducing the handshaking times
- Overlapping communication and computation

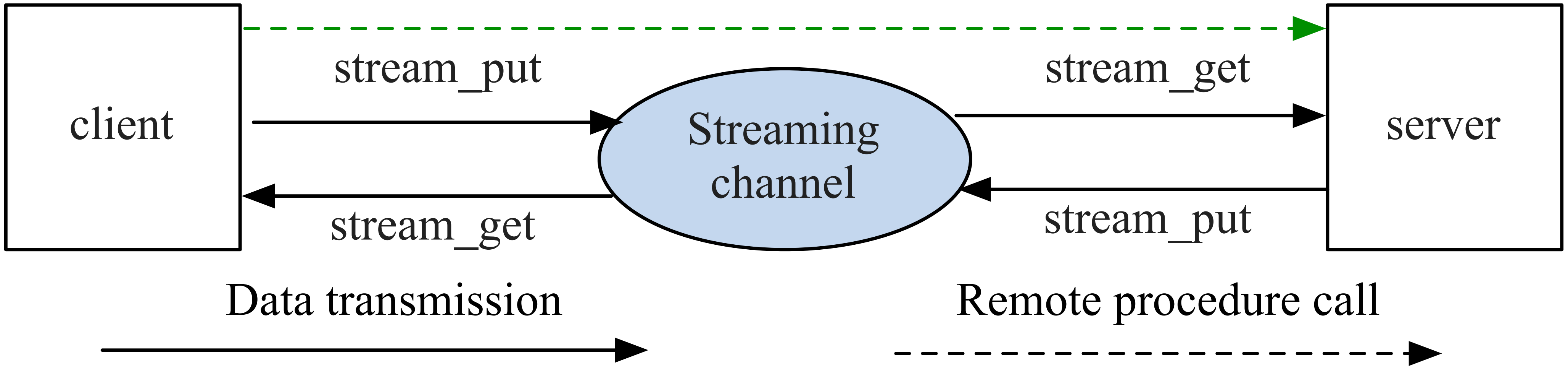


Few signal passing for internal handshaking

Software Framework of Streaming RPC

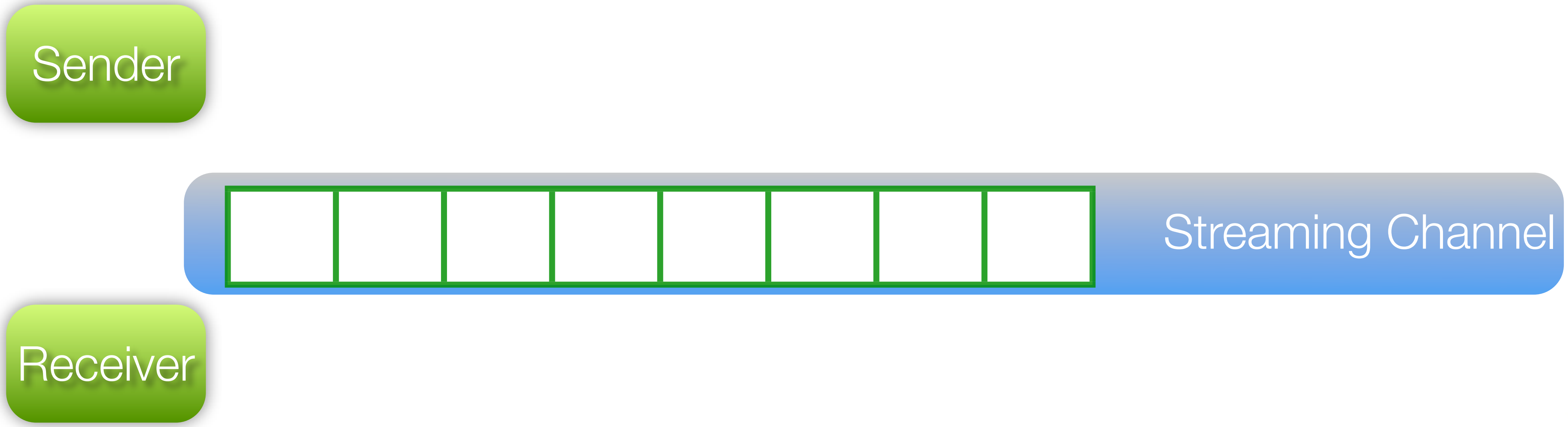
- Key components
  - Streaming channel
    - Automatically transmit data to the remote side
    - Abstraction for data streaming
  - Streaming buffer
    - Associated to a streaming channel
    - Providing data buffering
  - Stream controller
    - Monitoring and managing the streaming channel





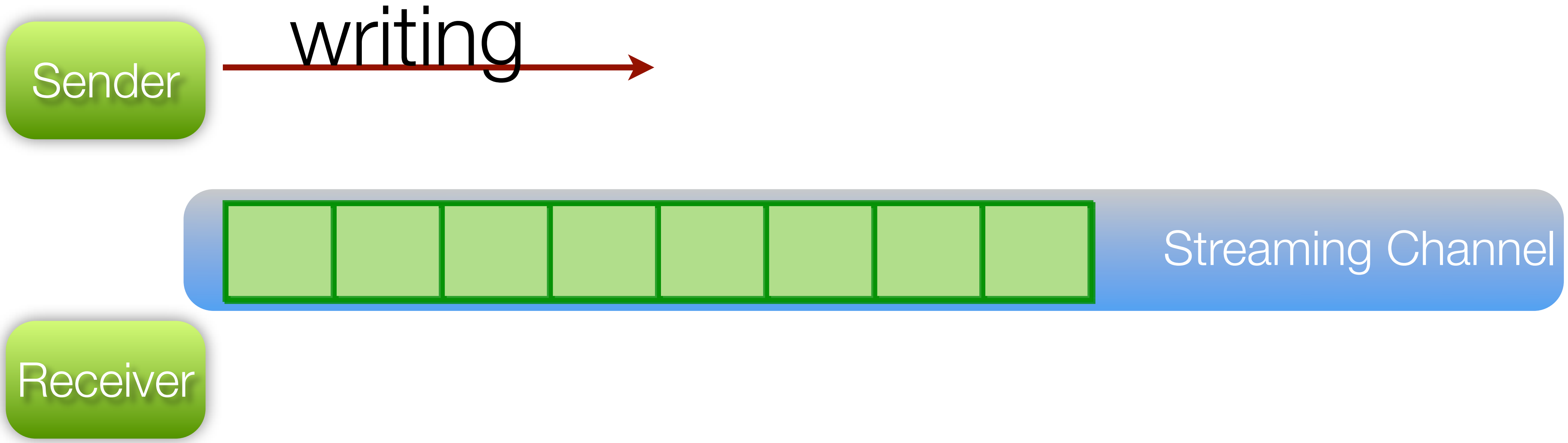
Asynchronous Communication Model

- Asynchronous RPC to avoid call-and-wait
- Data-driven model
- The stream controller first checks if a streaming buffer is ready
  - YES: start transmitting data
  - NO: suspends the sender/receiver until the streaming buffer is ready



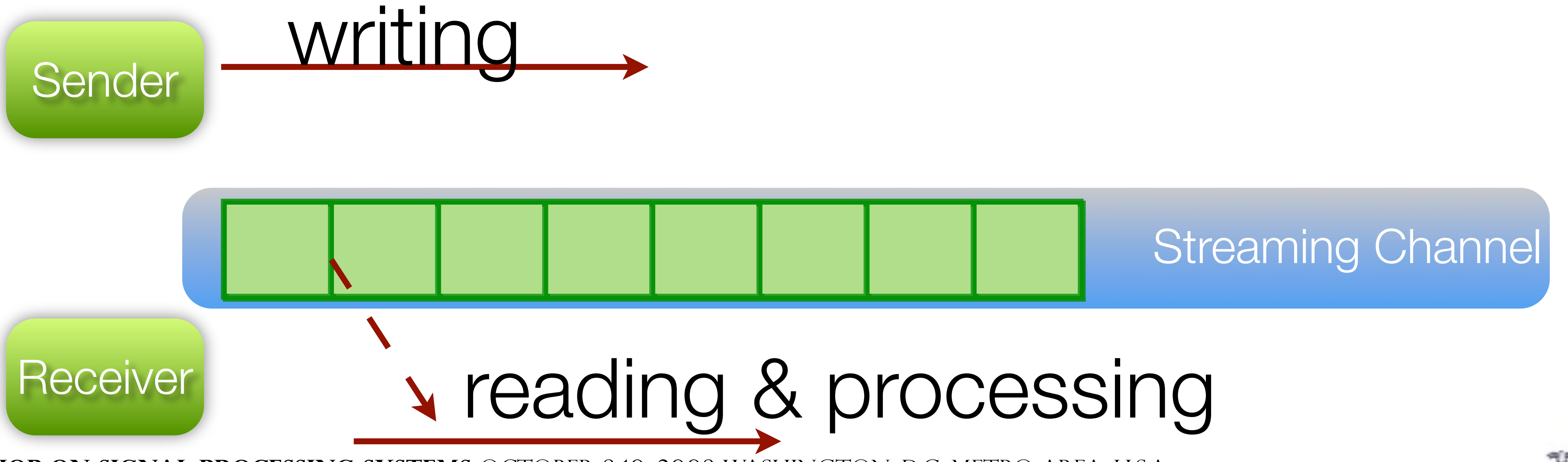
Asynchronous Communication Model

- Asynchronous RPC to avoid call-and-wait
- Data-driven model
- The stream controller first checks if a streaming buffer is ready
  - YES: start transmitting data
  - NO: suspends the sender/receiver until the streaming buffer is ready



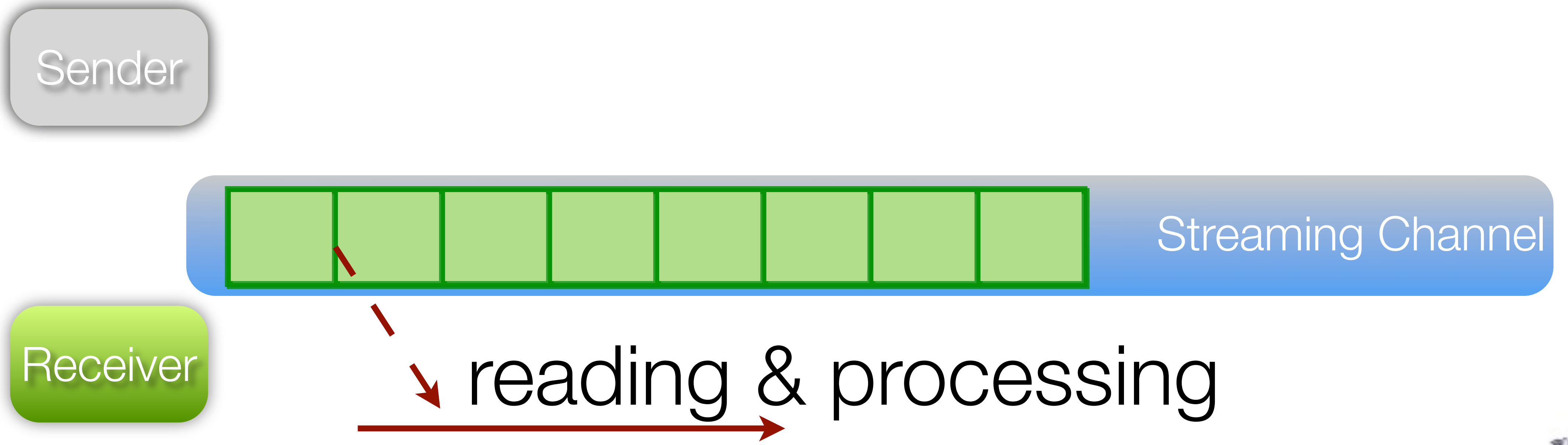
Asynchronous Communication Model

- Asynchronous RPC to avoid call-and-wait
- Data-driven model
- The stream controller first checks if a streaming buffer is ready
    - YES: start transmitting data
    - NO: suspends the sender/receiver until the streaming buffer is ready



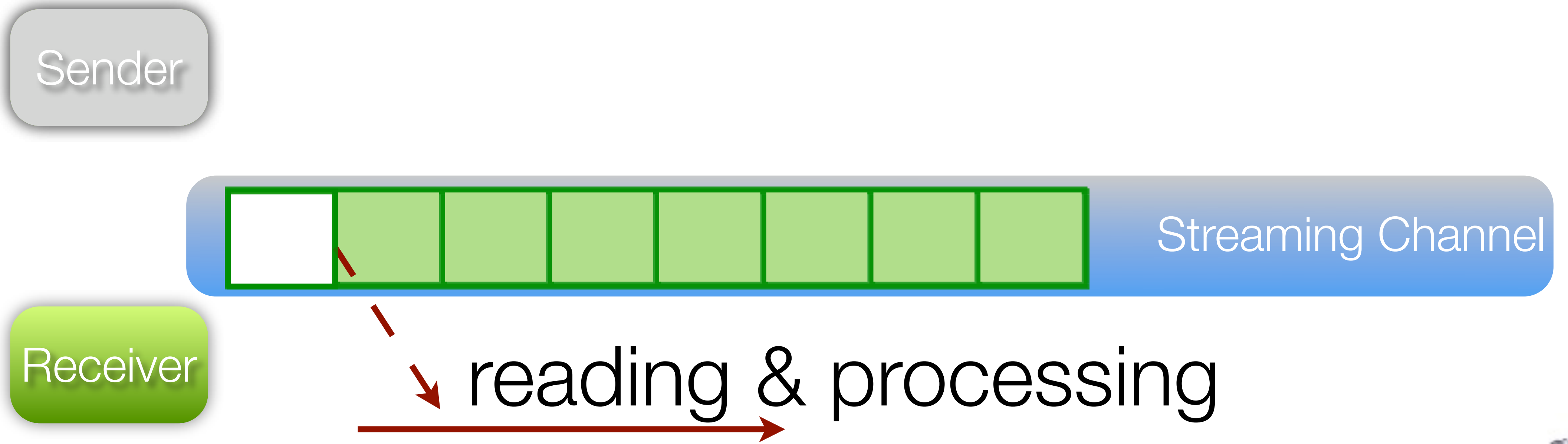
Asynchronous Communication Model

- Asynchronous RPC to avoid call-and-wait
- Data-driven model
- The stream controller first checks if a streaming buffer is ready
  - YES: start transmitting data
  - NO: suspends the sender/receiver until the streaming buffer is ready



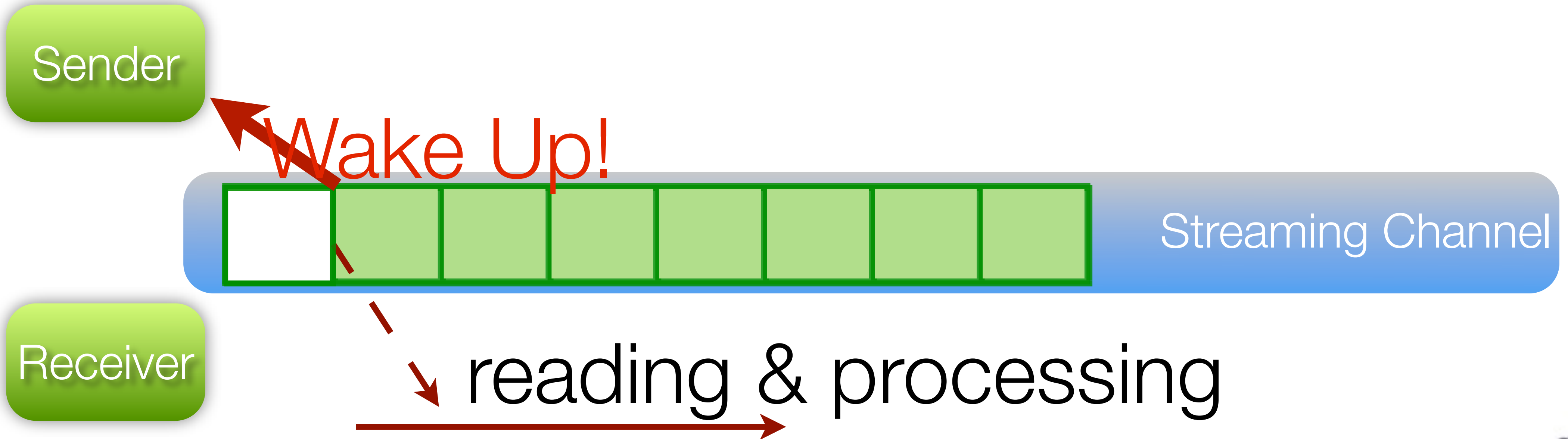
Asynchronous Communication Model

- Asynchronous RPC to avoid call-and-wait
- Data-driven model
- The stream controller first checks if a streaming buffer is ready
  - YES: start transmitting data
  - NO: suspends the sender/receiver until the streaming buffer is ready



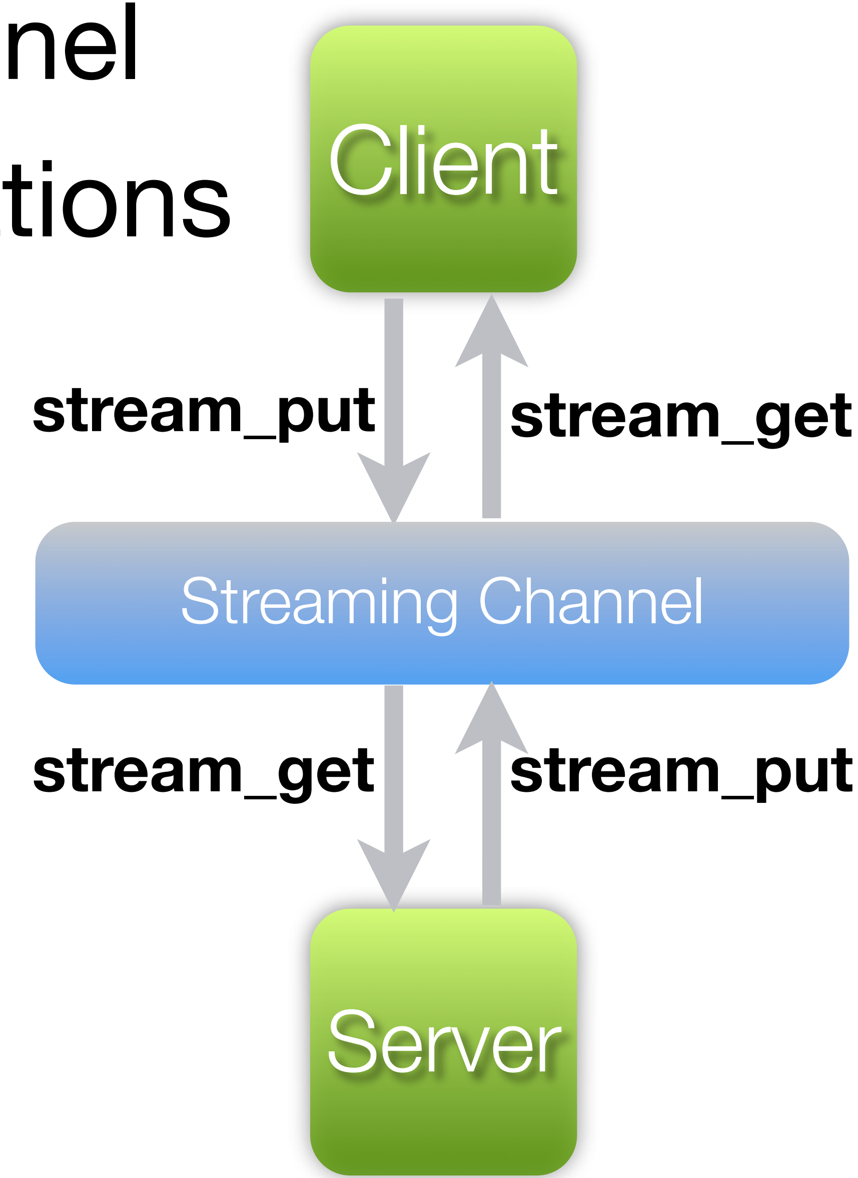
Asynchronous Communication Model

- Asynchronous RPC to avoid call-and-wait
- Data-driven model
- The stream controller first checks if a streaming buffer is ready
  - YES: start transmitting data
  - NO: suspends the sender/receiver until the streaming buffer is ready



Application Interfaces

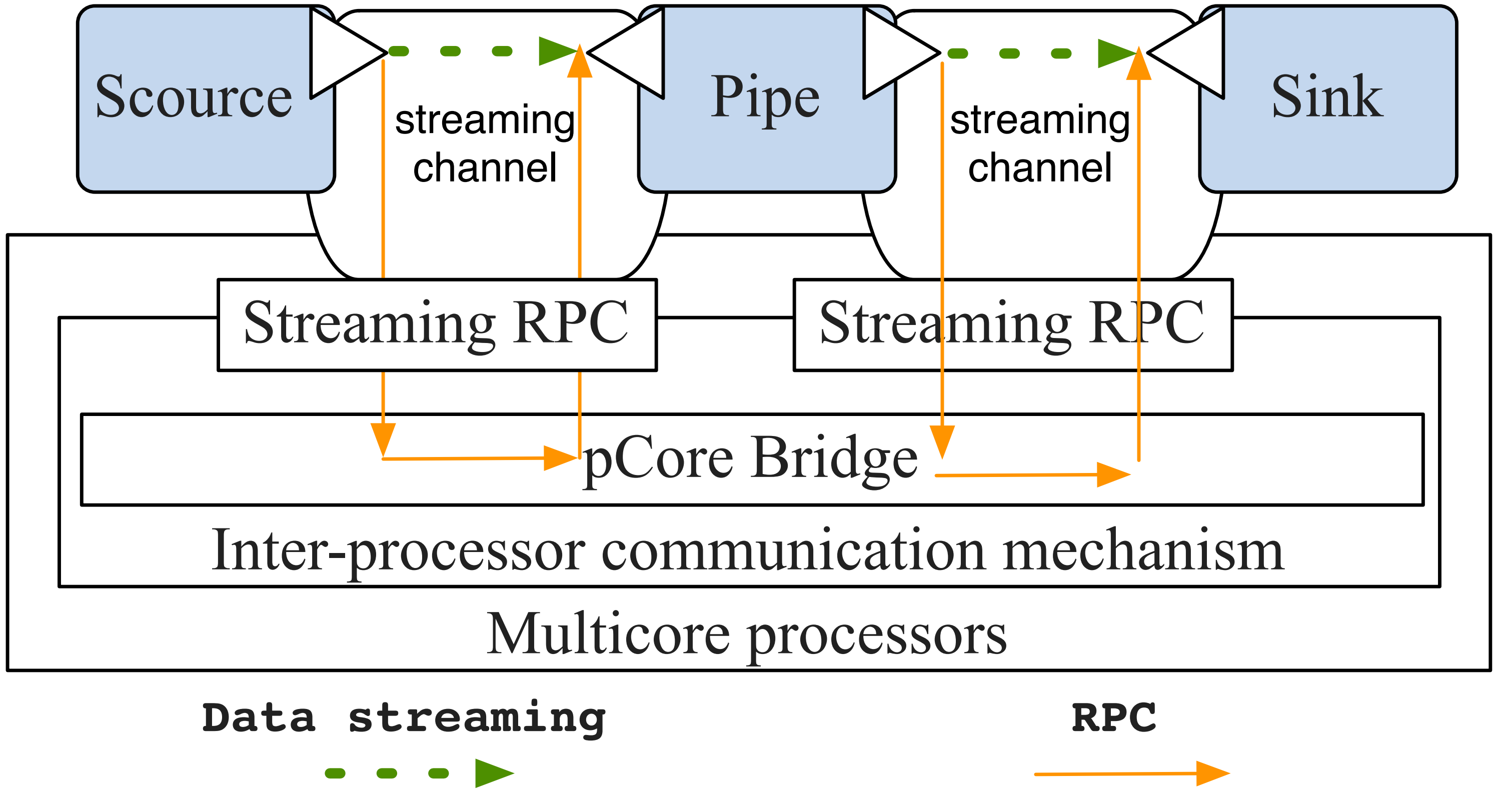
- An RPC is associated with streaming channels
- The client and server can send/get data to/from the channel
- Streaming operations
  - stream_get
  - stream_put
  - stream_push
  - stream_pop
  - stream_create
  - stream_rpc



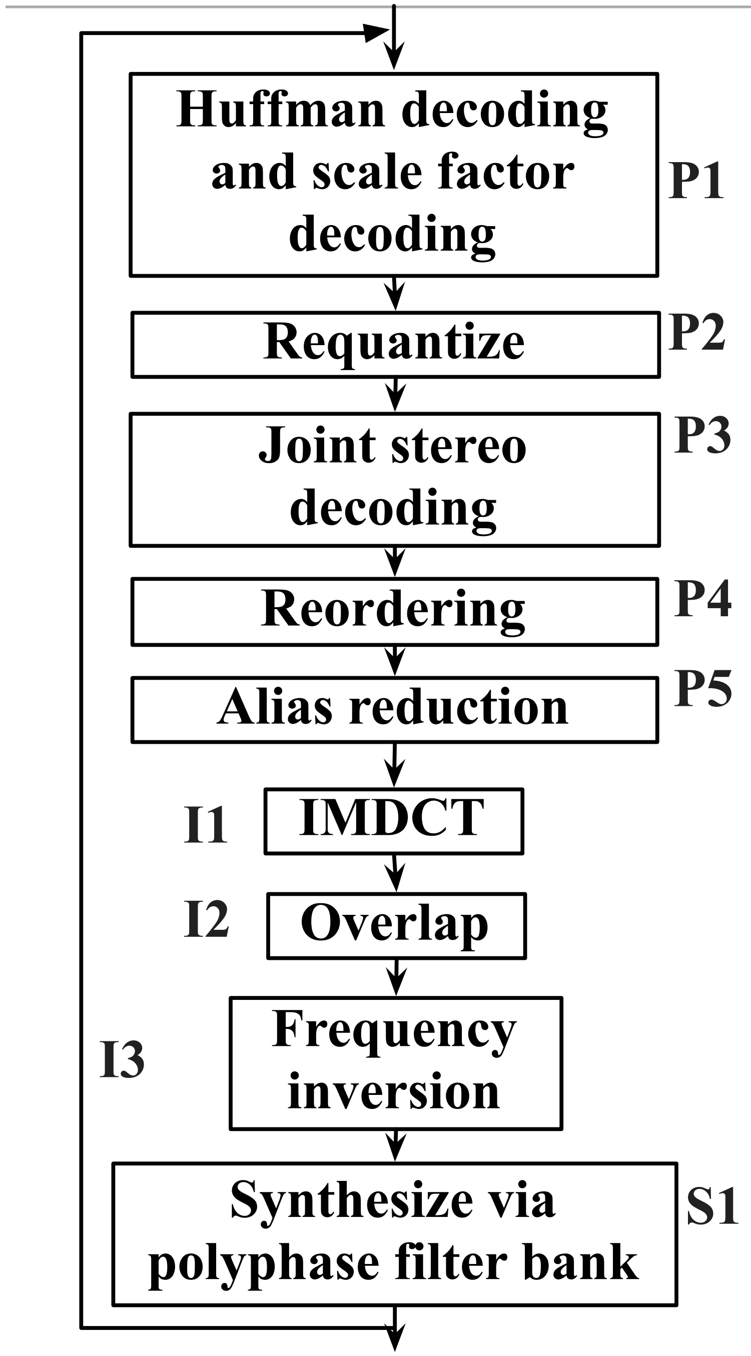
```
/* Streaming RPC client */
void MP3_decoder(){
        stream_rpc(_imdct_, _transmitter_);
}
void _transmitter_(){
        STREAM_ID id = 4;
        /* Initializing streaming channel */
        stream_create(id);
        /* Pushing data to streaming channel */
        stream_put(id, DATA);
        stream_push(id);
        ...
}
...
/* Streaming RPC server */
void _imdct_(){
        STREAM_ID id = 4;
        /* Initializing streaming channel */
        stream_create(id);
        /* Aggregating data from streaming
         channel */
        stream_get(id, DATA);
        stream_pop(id);
        ...
}
...
```

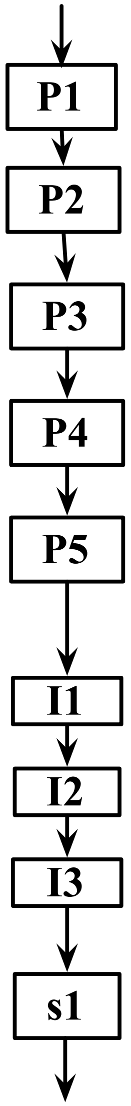
Basic Software Components

- An application is composed of three basic structural components
  - **Source**: retrieves data and dispatch it to the remote process
  - **Pipe**: serves as a computational unit
  - **Sink**: aggregates data for integration

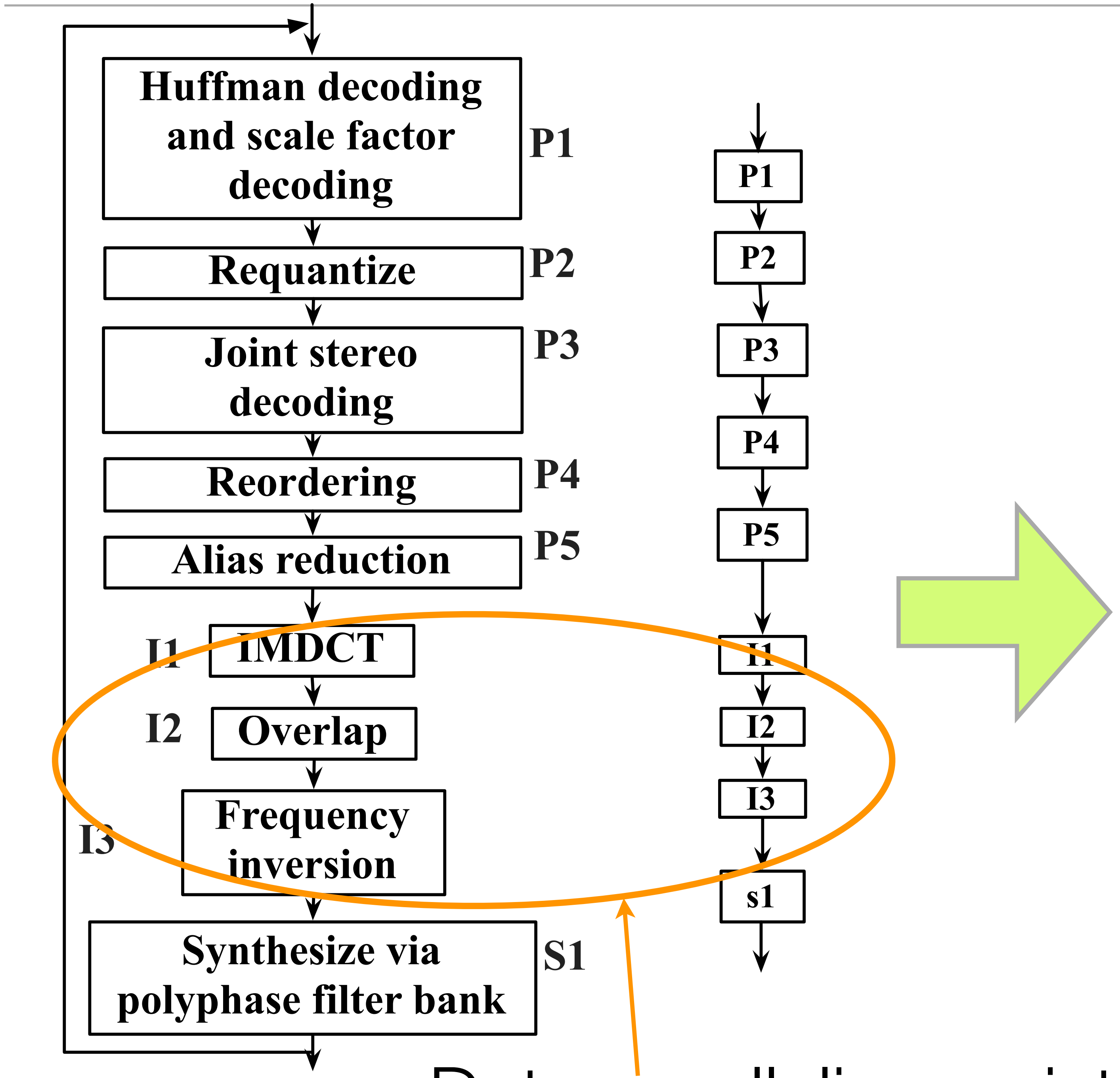


Example: MP3 Decoder





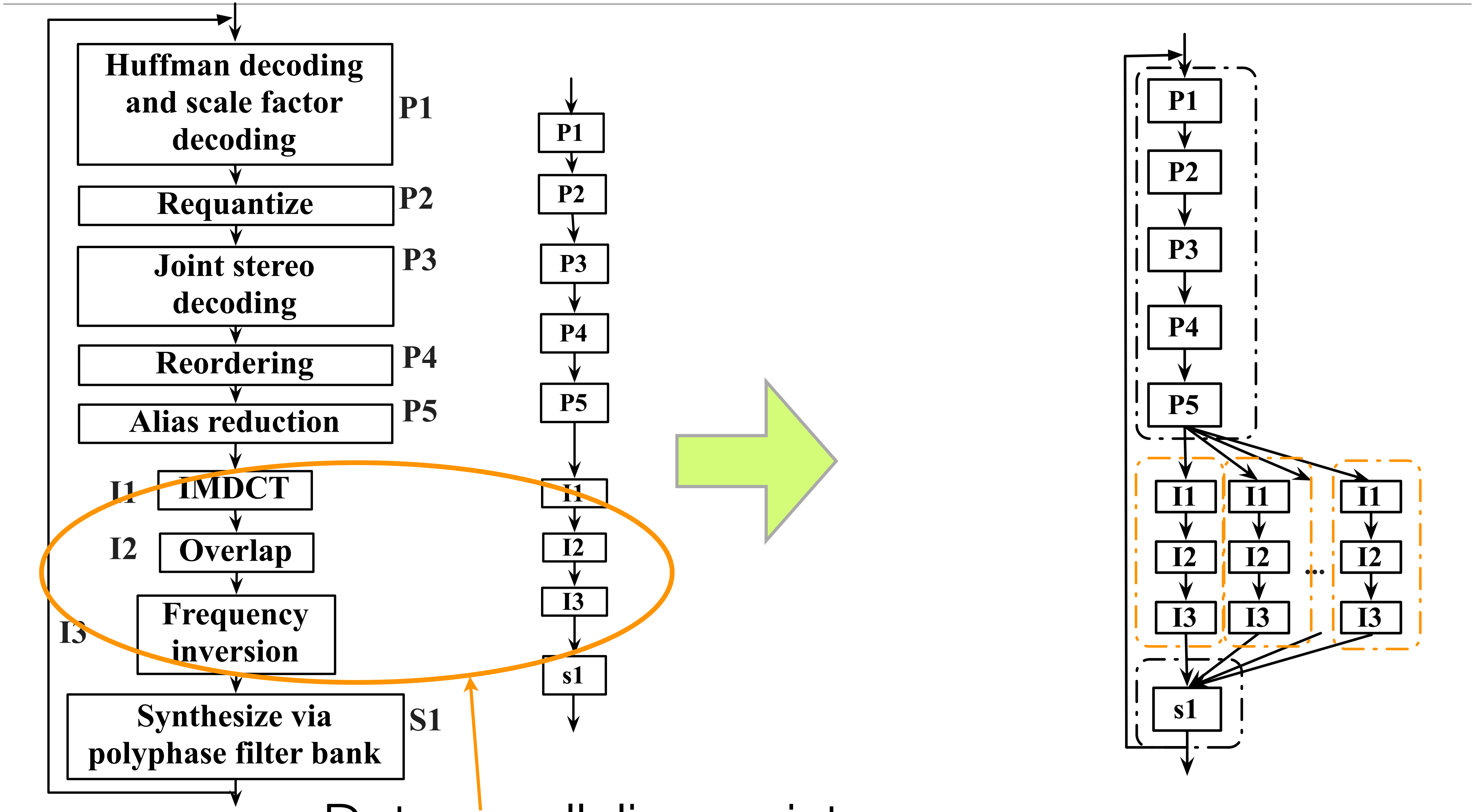
Example: MP3 Decoder



Data parallelism exists

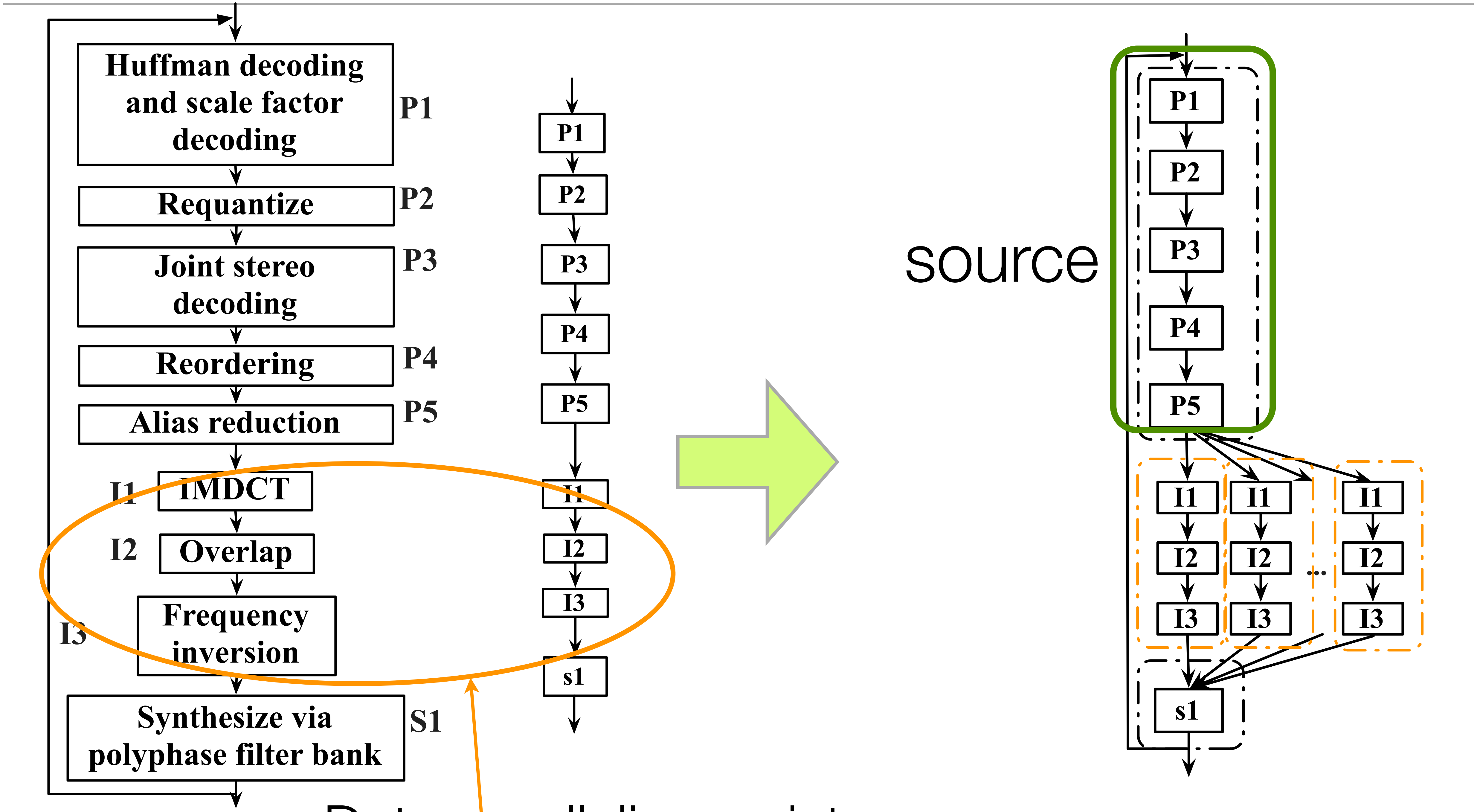
partitioned to SPUs

Example: MP3 Decoder



Data parallelism exists partitioned to SPUs

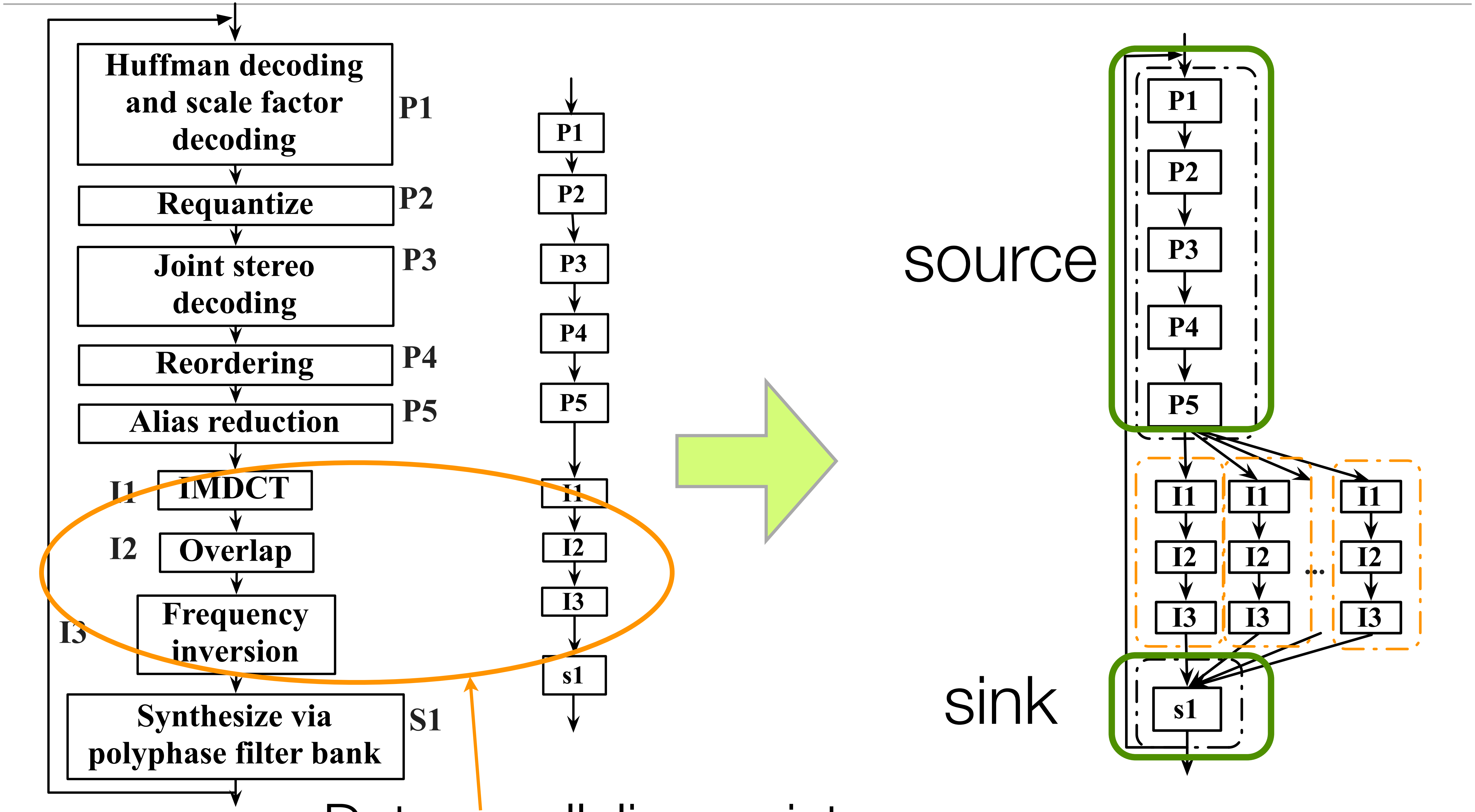
Example: MP3 Decoder



source

Data parallelism exists partitioned to SPUs

Example: MP3 Decoder



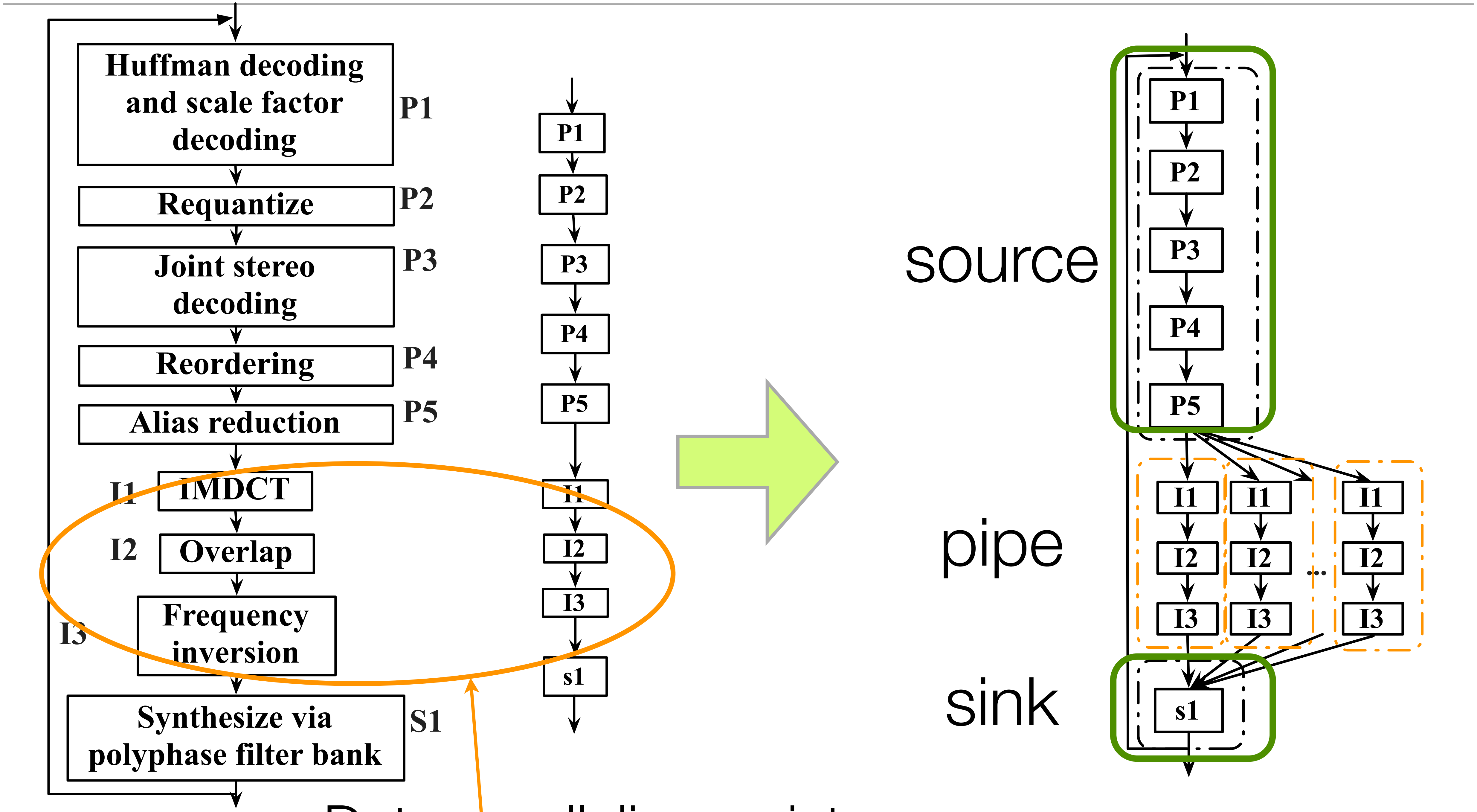
source

sink

Data parallelism exists

partitioned to SPUs

Example: MP3 Decoder

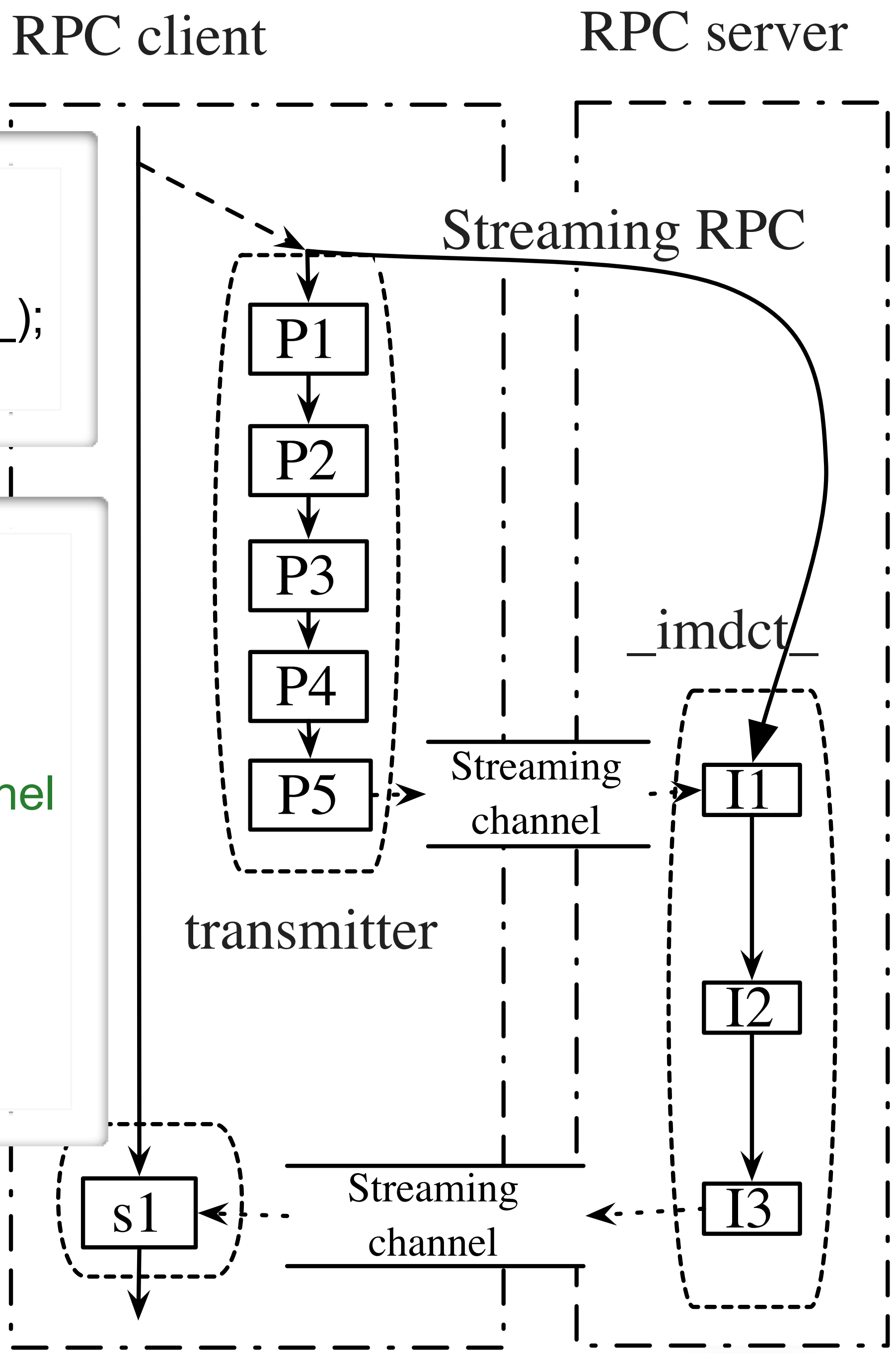


Data parallelism exists partitioned to SPUs

Sample Code of MP3 Decoder

```
/* Streaming RPC client */
void MP3_decoder(){
      stream_rpc(_imdct_, _transmitter_);
}
```

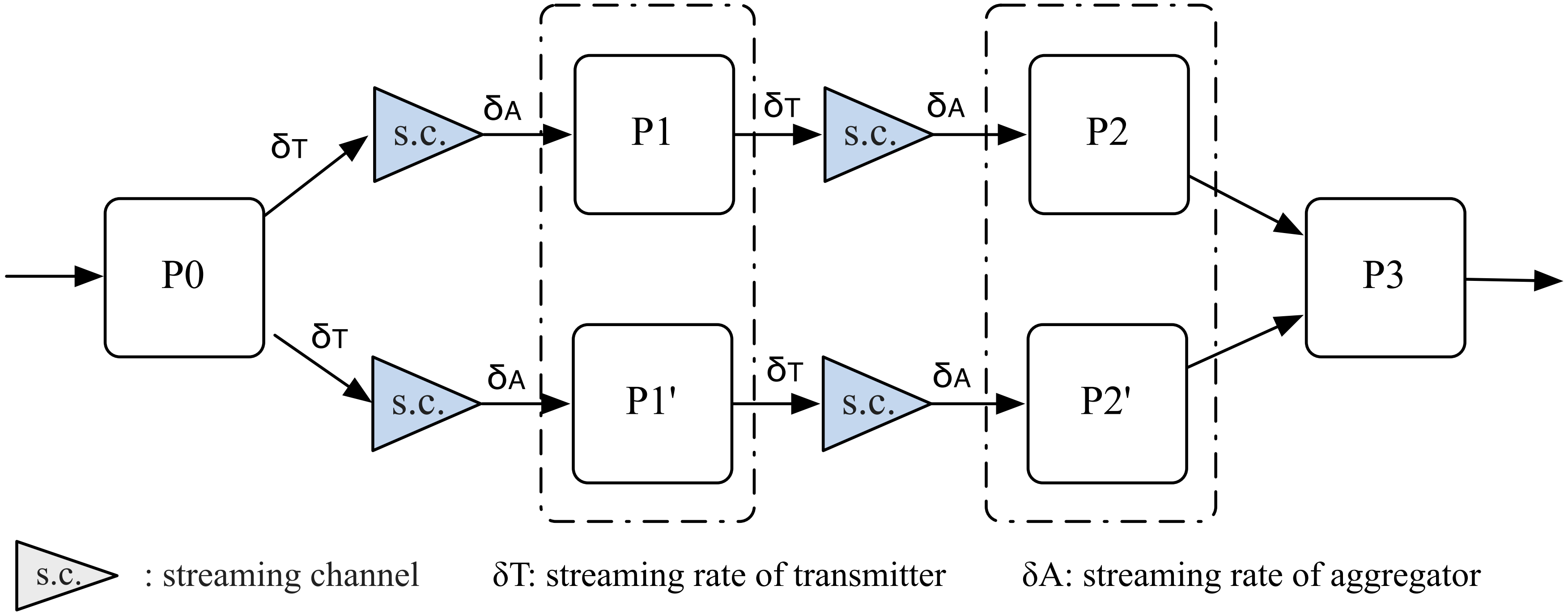
```
void _transmitter_(){
      STREAM_ID id = 4;
      /* Initializing streaming channel */
      stream_create(id);
      /* Pushing data to streaming channel
*/
      stream_put(id, DATA);
      stream_push(id);
      ...
}
```



```
/* Streaming RPC server */
void _imdct_(){
      STREAM_ID id = 4;
      /* Initializing streaming channel
*/
      stream_create(id);
      /* Aggregating data from
streaming
       channel */
      stream_get(id, DATA);
      stream_pop(id);
      ...
}
```

Streaming Rate (δ)

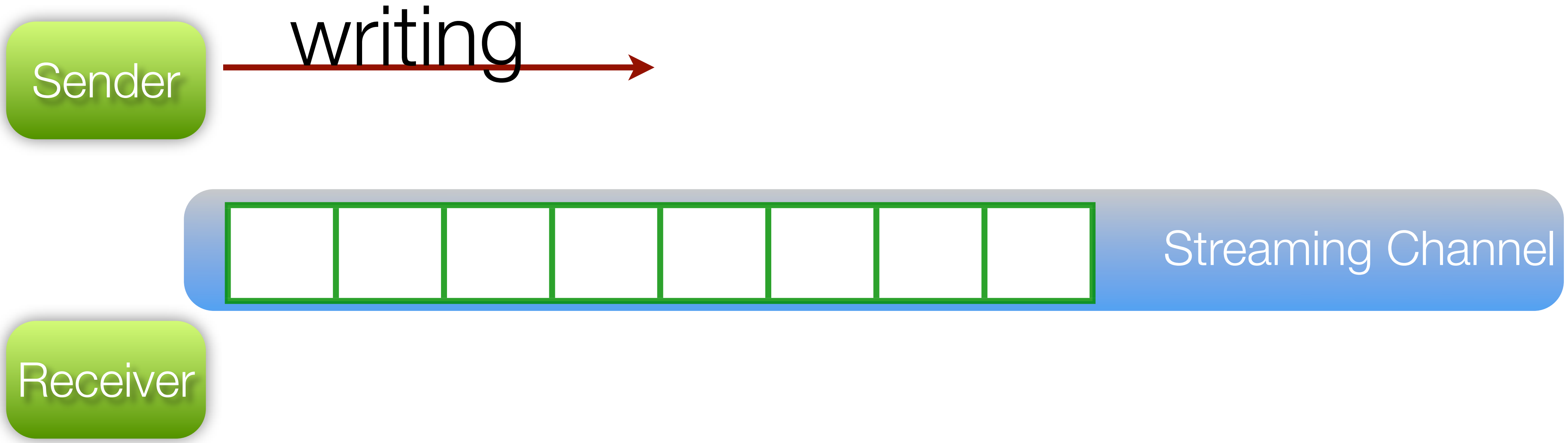
- Streaming rate: amount of streaming data accessed by the sender/receiver per unit of time
- Difference in I/O latency, processing speed, and computation workloads result in asymmetry in streaming rate between processors
- Result in frequent suspension and waking up!
  - Increasing amount of implicit internal RPC handshaking times
  - Ex. when $\boldsymbol{\delta_A} > \boldsymbol{\delta_T}$, the receiver is suspended frequently



: streaming channel δT: streaming rate of transmitter δA: streaming rate of aggregator

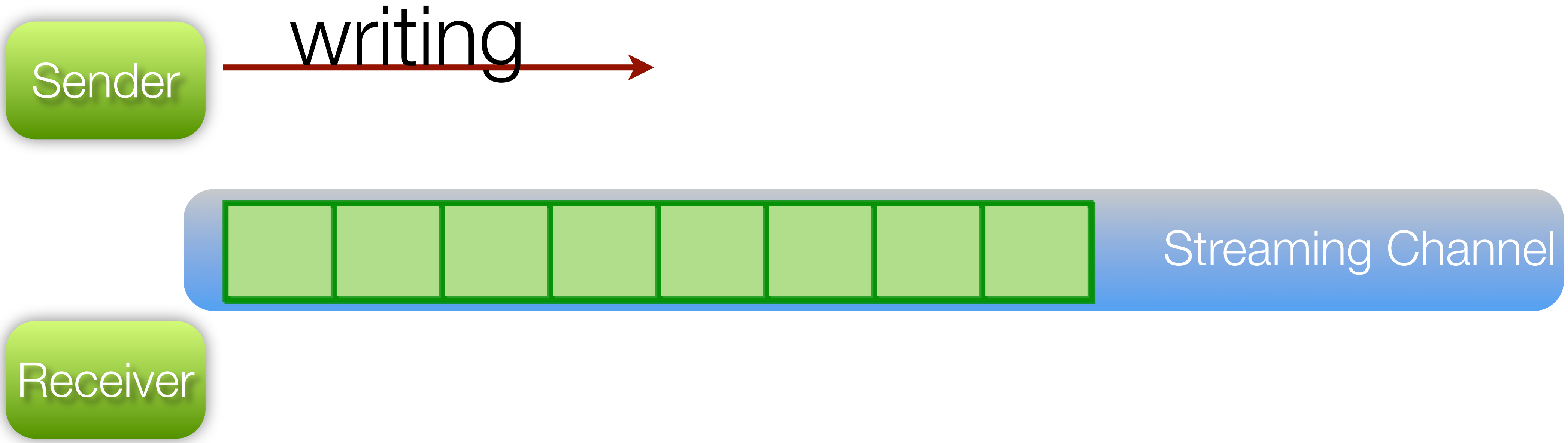
Setting Threshold Number

- To avoid frequent suspension and waking up!
- Assigning a threshold value to a streaming channel
  - The stream controller only wakes up the sender/receiver when a streaming channel satisfies the threshold criterion
  - ex. threshold value = 4



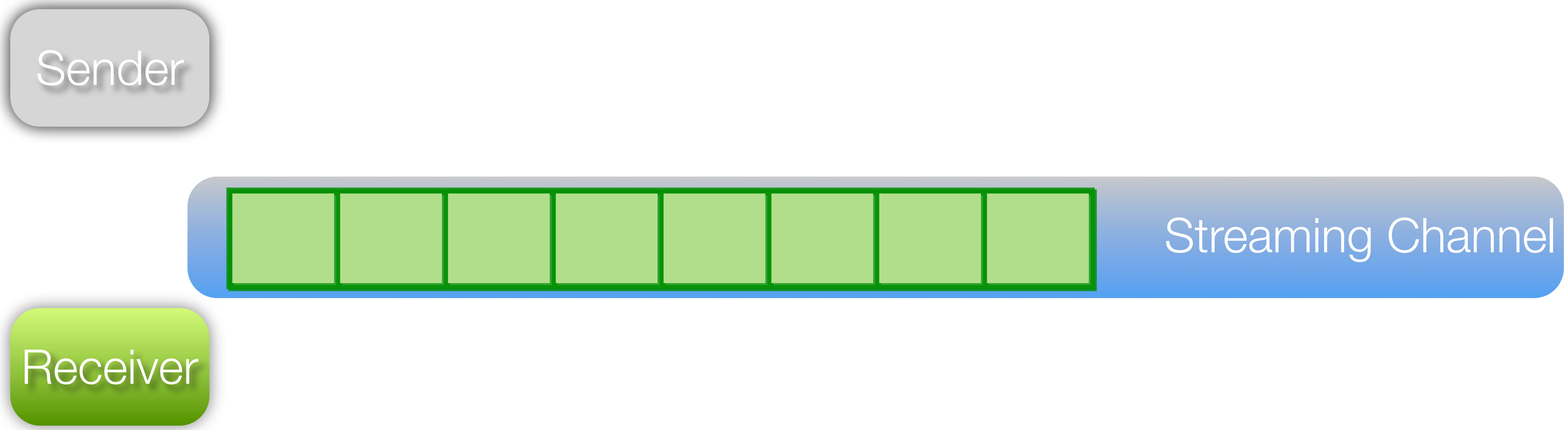
Setting Threshold Number

- To avoid frequent suspension and waking up!
- Assigning a threshold value to a streaming channel
  - The stream controller only wakes up the sender/receiver when a streaming channel satisfies the threshold criterion
  - ex. threshold value = 4



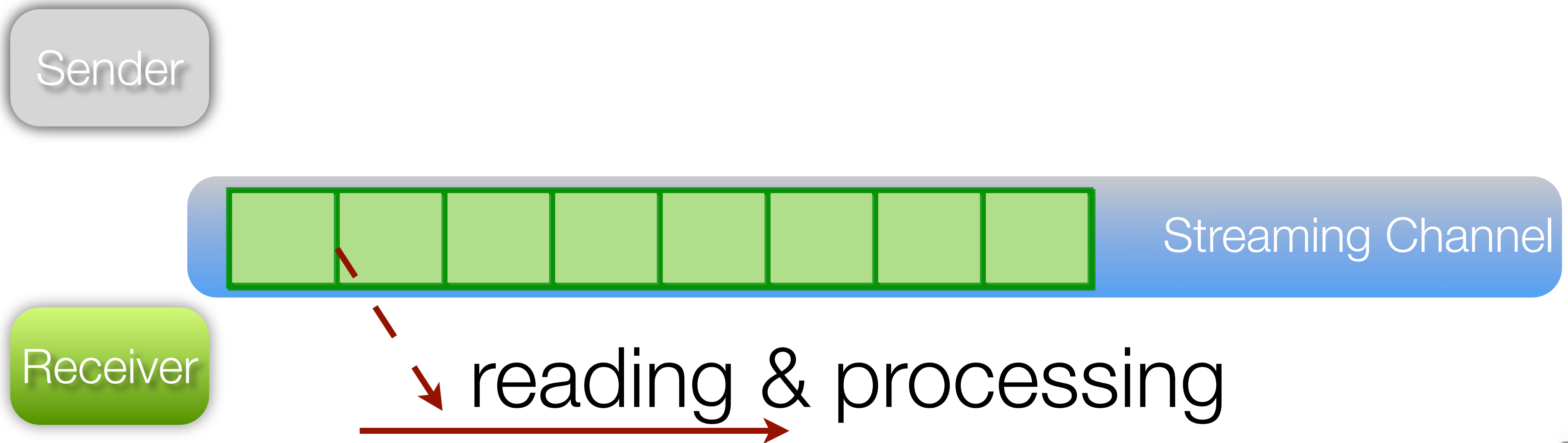
Setting Threshold Number

- To avoid frequent suspension and waking up!
- Assigning a threshold value to a streaming channel
  - The stream controller only wakes up the sender/receiver when a streaming channel satisfies the threshold criterion
  - ex. threshold value = 4



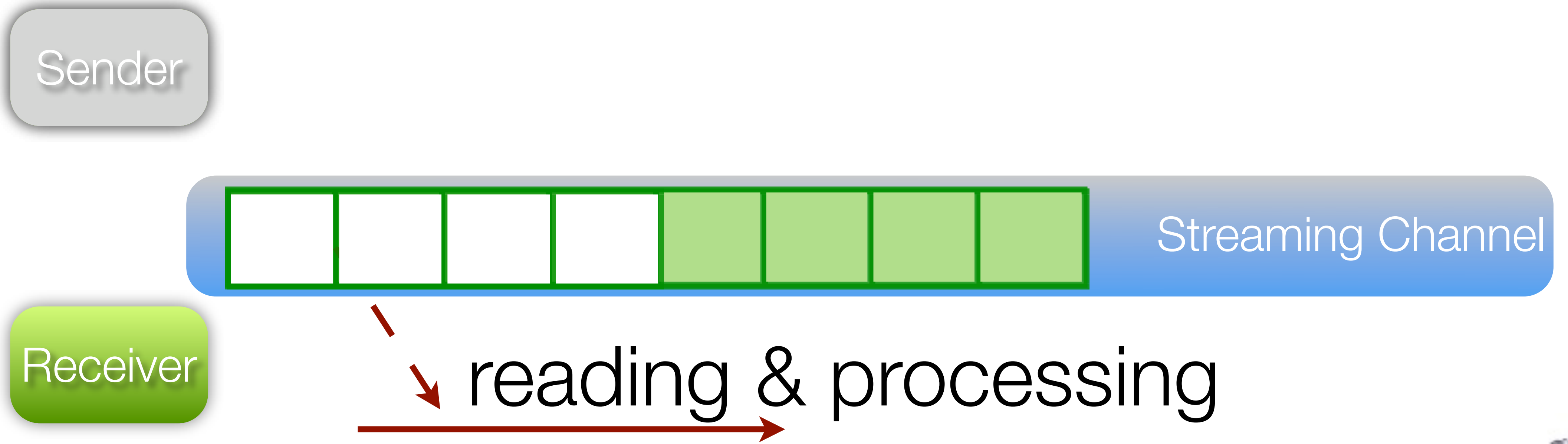
Setting Threshold Number

- To avoid frequent suspension and waking up!
- Assigning a threshold value to a streaming channel
  - The stream controller only wakes up the sender/receiver when a streaming channel satisfies the threshold criterion
  - ex. threshold value = 4



Setting Threshold Number

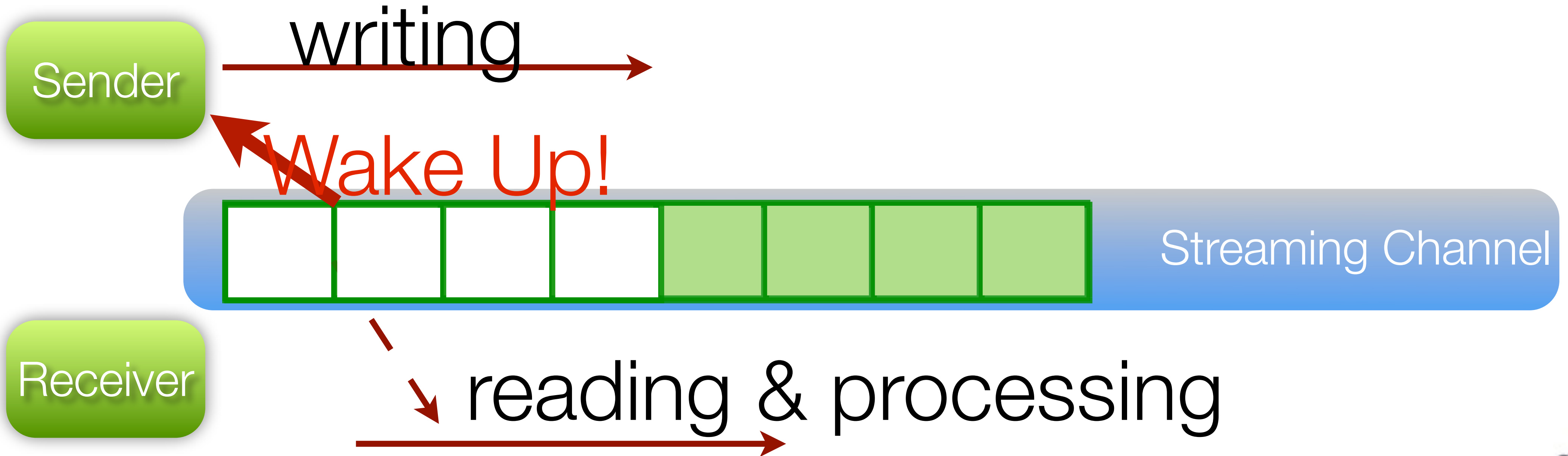
- To avoid frequent suspension and waking up!
- Assigning a threshold value to a streaming channel
  - The stream controller only wakes up the sender/receiver when a streaming channel satisfies the threshold criterion
  - ex. threshold value = 4



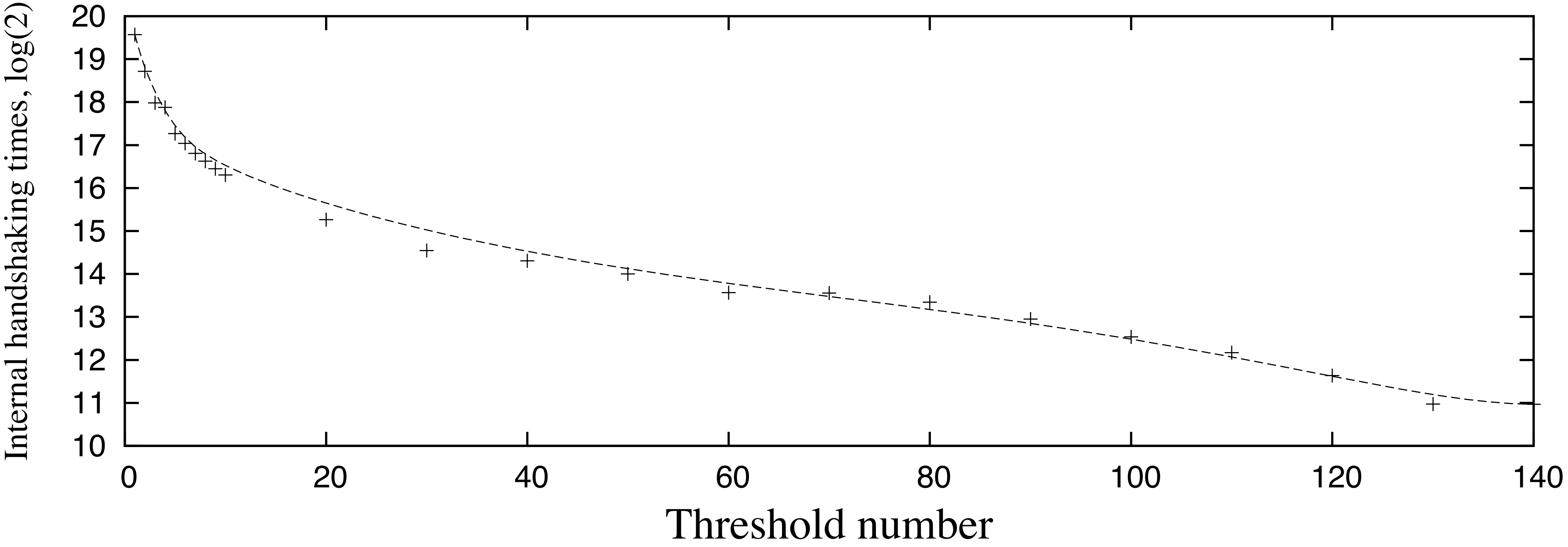
reading & processing

Setting Threshold Number

- To avoid frequent suspension and waking up!
- Assigning a threshold value to a streaming channel
  - The stream controller only wakes up the sender/receiver when a streaming channel satisfies the threshold criterion
  - ex. threshold value = 4



Effects of Setting Threshold Number



- How to decide the threshold number for a channel to reduce the internal handshaking times?
- Optimal solution for reducing handshaking times is ∞ !!
  - Memory is limited and valuable in embedded system
  - A response time requirement for multimedia applications

Analytic Model for Deciding Threshold n

- To meet the response time constraint of the application
- Time of the first element (Δbytes, in bandwidth **B**)to be processed after waiting for the sender to transmitting **n** stream elements must be less than the timing constraint

Response time constrain

Time required for transferring **n** streaming elements

$$T_r \geq o_t + \frac{n}{\delta_T} + l + \frac{\triangle}{B}$$

Time required for receiver to get the first stream element

Overhead of triggering the remote process

Time required for receiver to process the first stream element

thus,

$$n = \lfloor (T_r - o_t - l - \frac{\triangle}{B}) \times \delta_T \rfloor$$

Experiment Platform

PAC Evaluation Board:

- **P**arallel **A**rchitecture **C**ore **(PAC),** Developed by STC, ITRI, Taiwan
- ARM9 (300MHz)
- PAC DSP (250 MHz,5-way issued VLIW)
  - 64 KB data memory
  - 32 KB instruction cache
- Linux 2.6.17
- pCore Bridge(*) communication module

OMAP 5912 OSK :

- Developed by TI
- ARM9 (192 MHz)
- TMS320 C55x
- Linux 2.6.17
- pCore Bridge(*) communication module

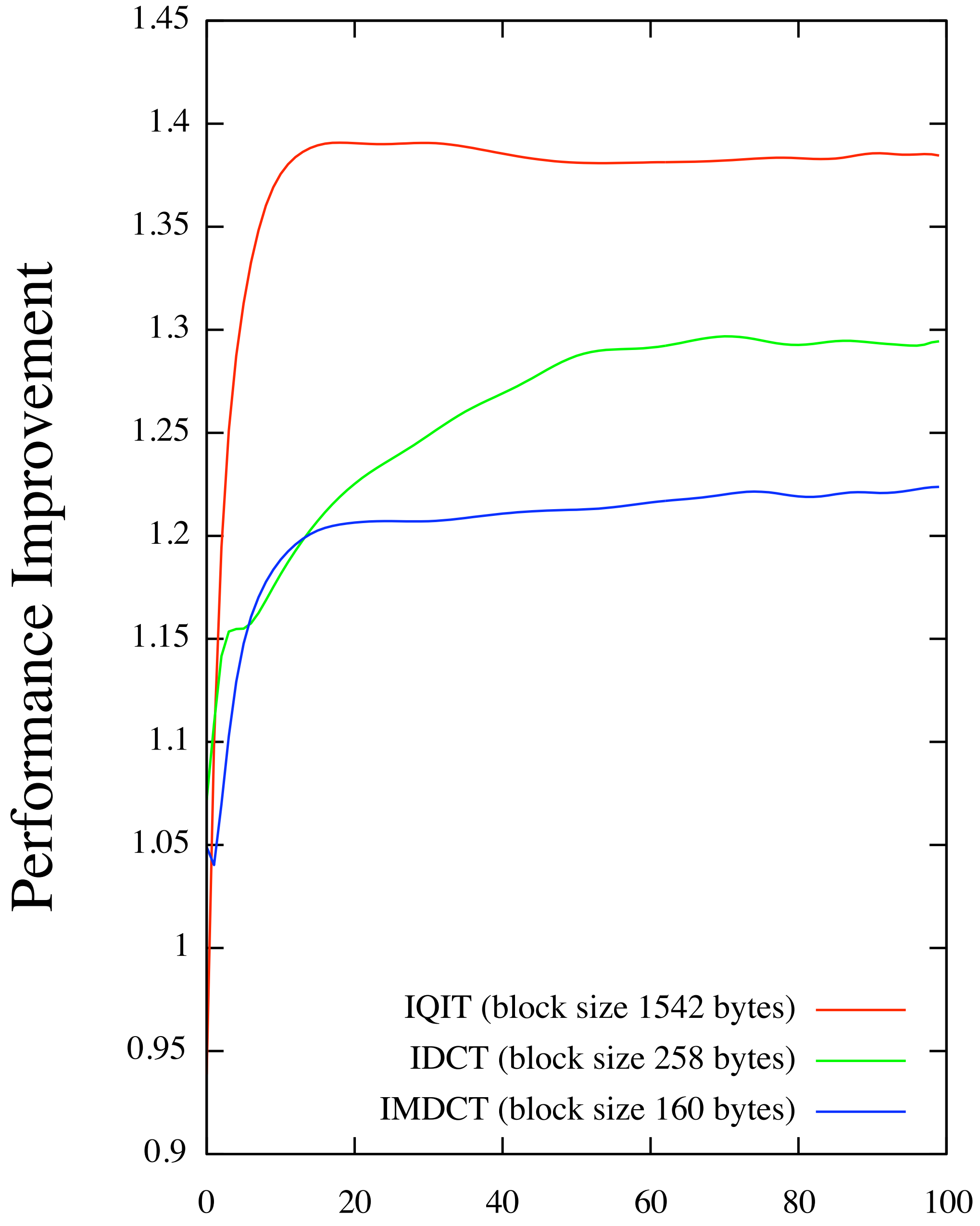
* Hsieh, K., Lin, Y., Huang, C., and Lee, J. 2008. Enhancing Microkernel Performance on VLIW DSP Processors via Multiset Context Switch. *J. Signal Process. Syst*.

Experiment Setup

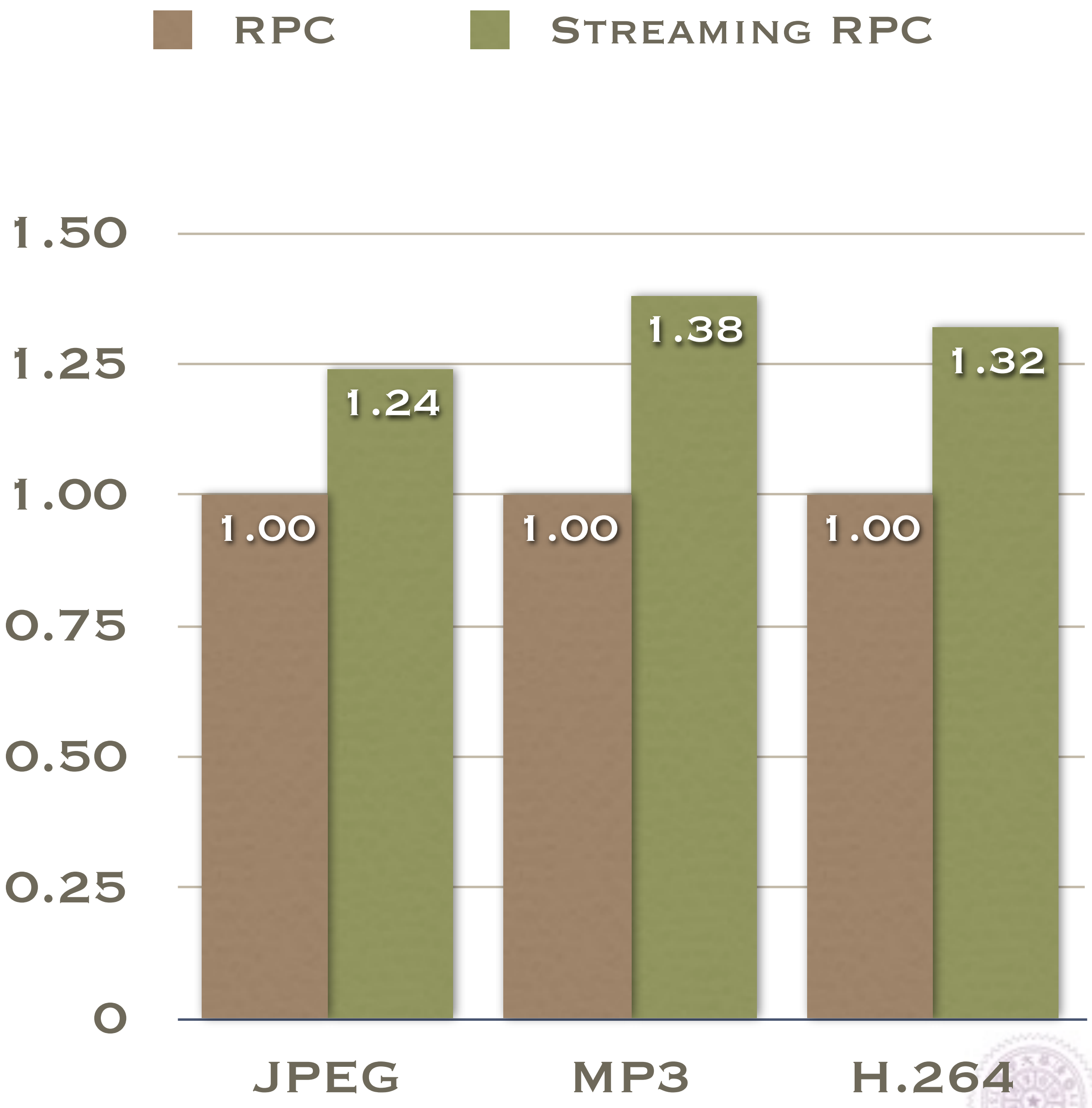
- Three applications: JPEG, MP3, and H.264 decoders to demonstrate the performance improvement
- Three application kernels: IDCT, IMDCT, IQ/IT to show the characteristics of streaming RPC
- Effects of threshold value to response time and performance are also evaluated

Performance Improvement on PAC

Performance evaluation of different kernels

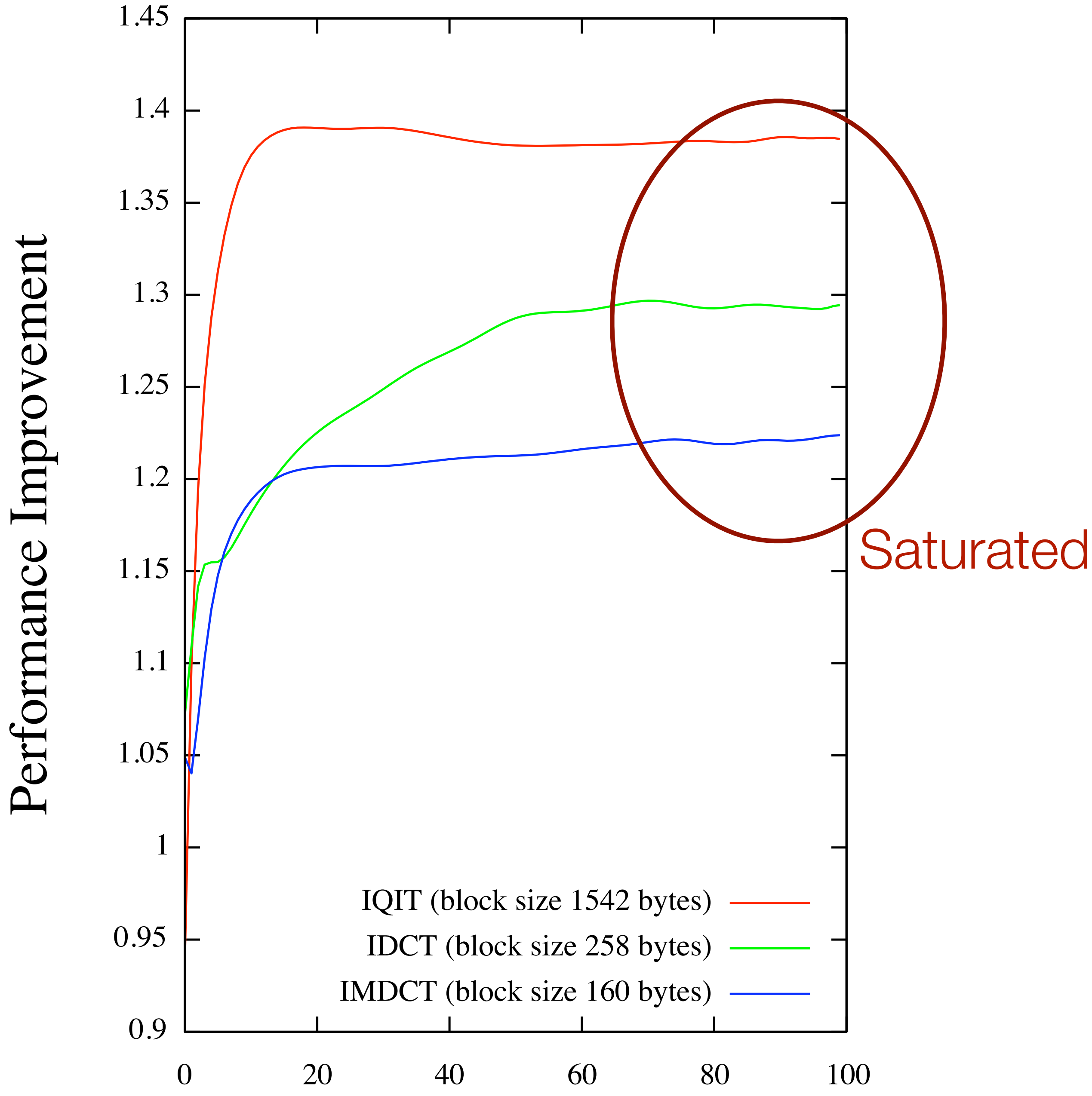


Performance improvement of applications

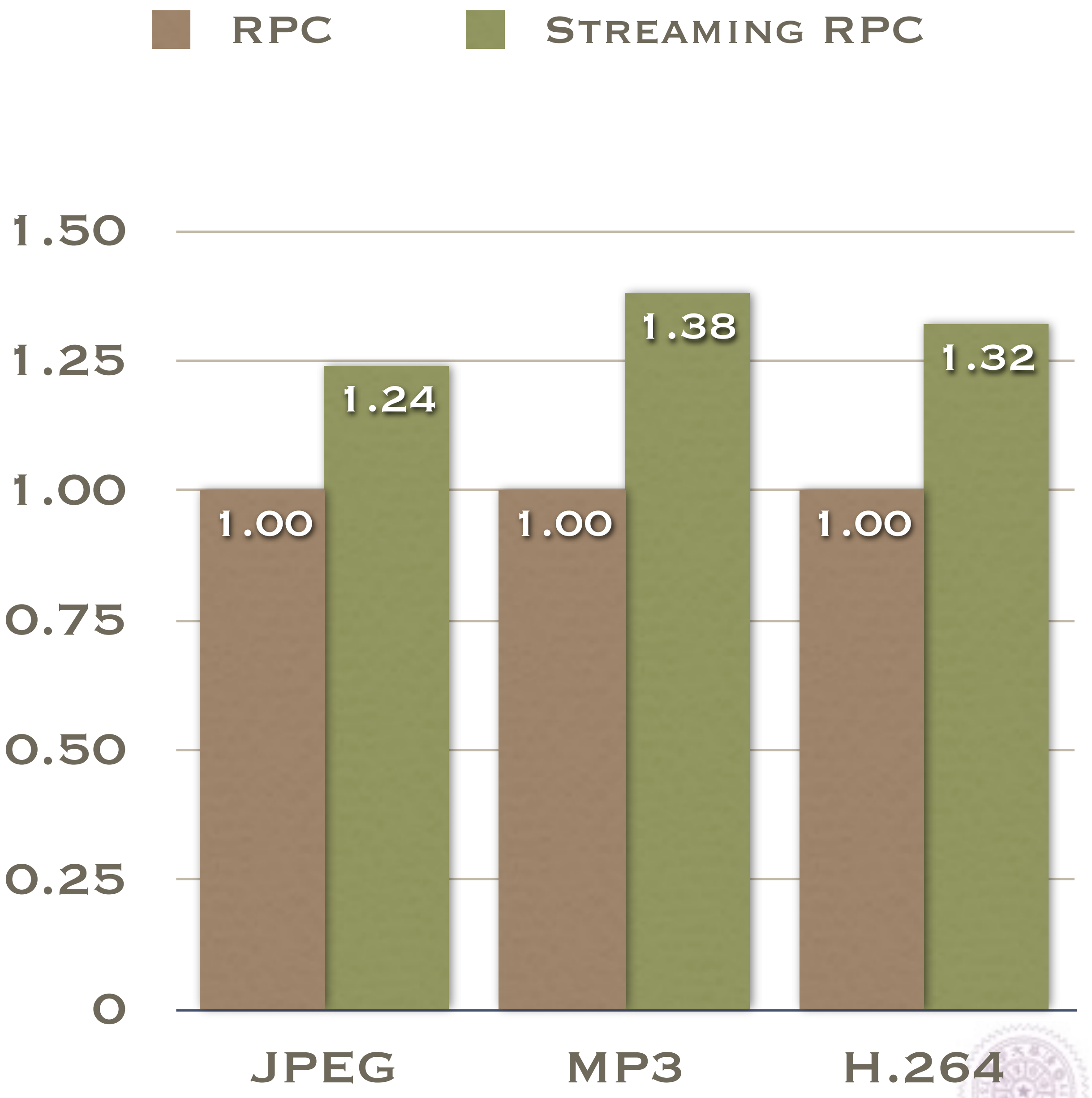


Performance Improvement on PAC

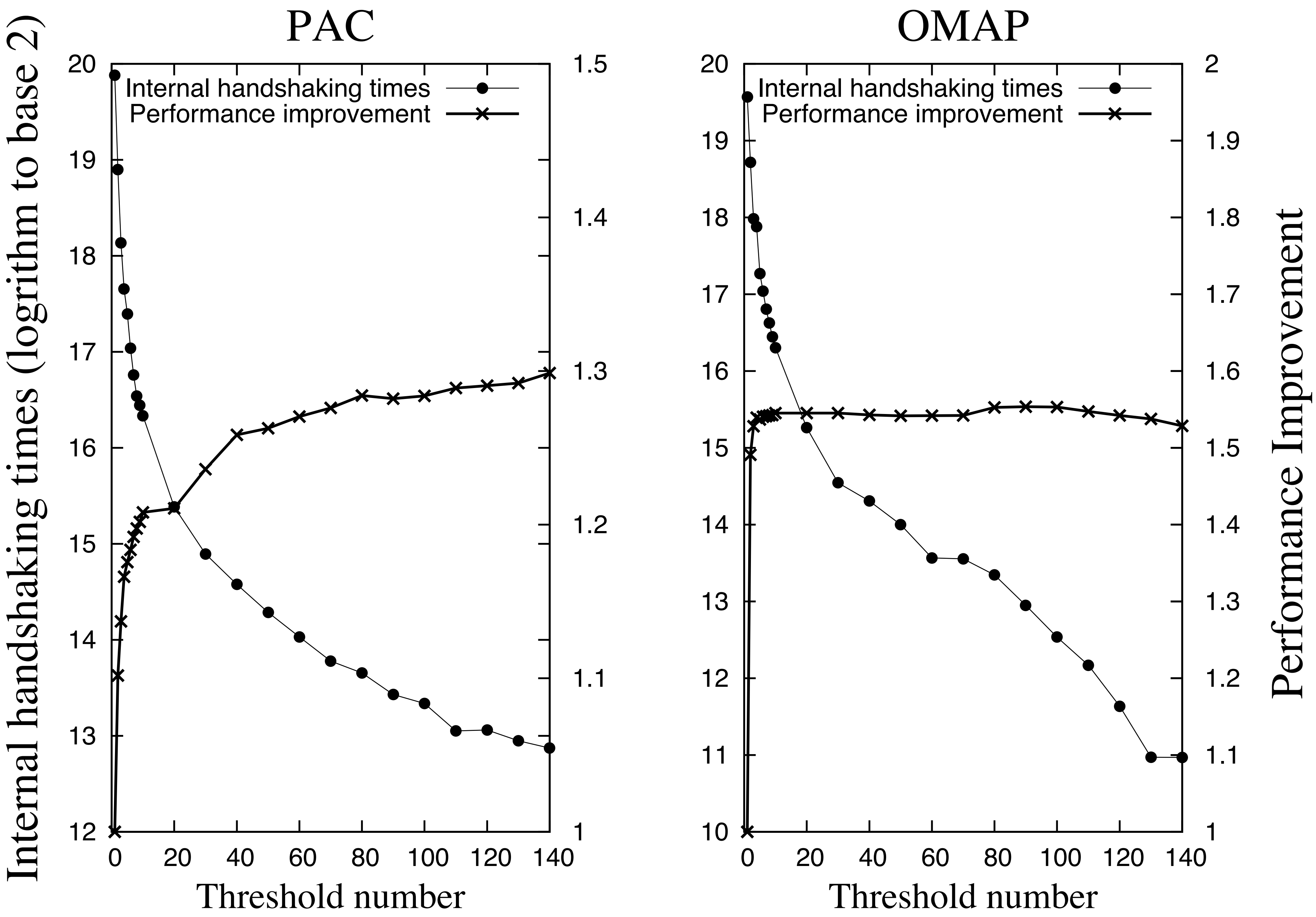
Performance evaluation of different kernels



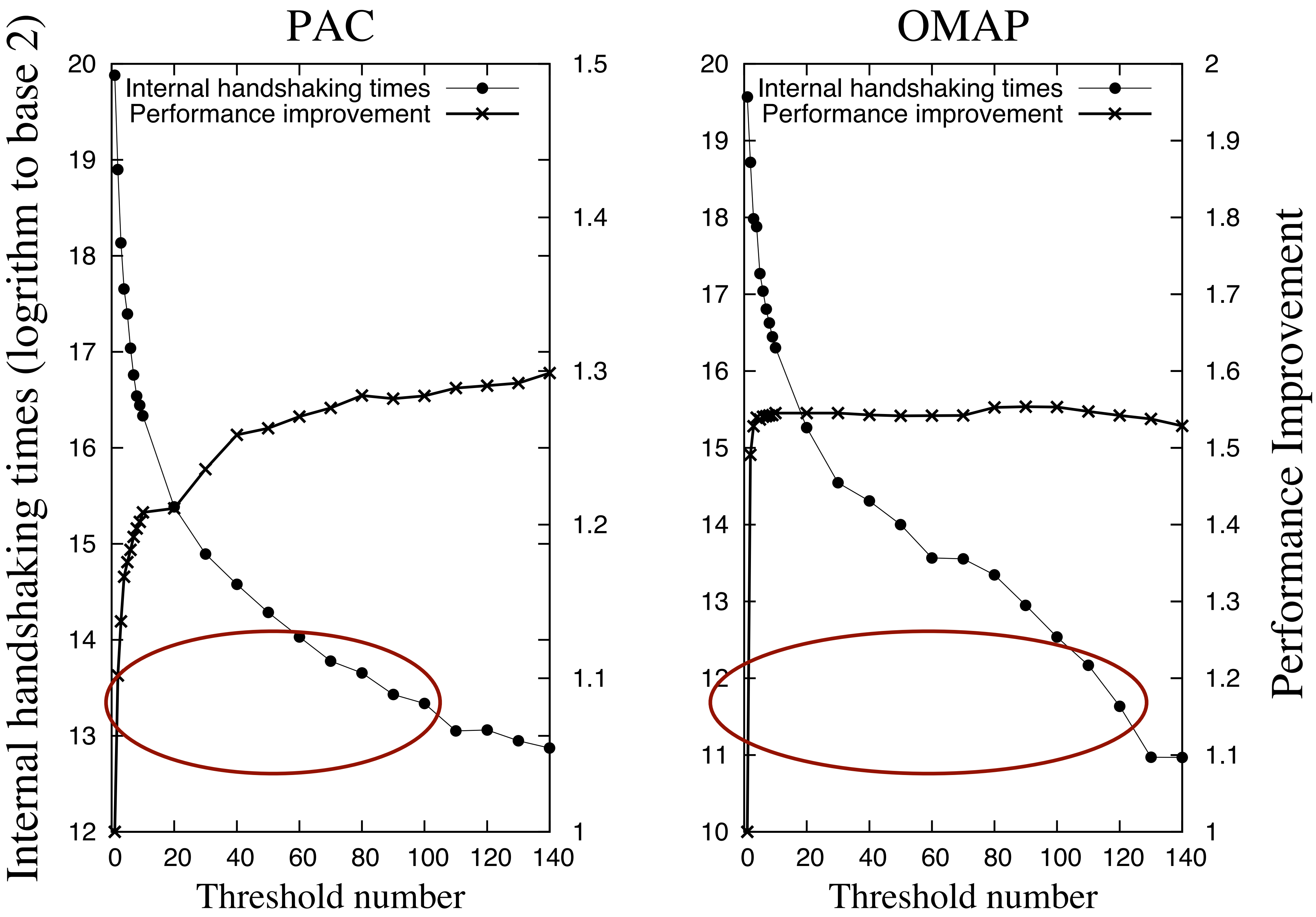
Performance improvement of applications



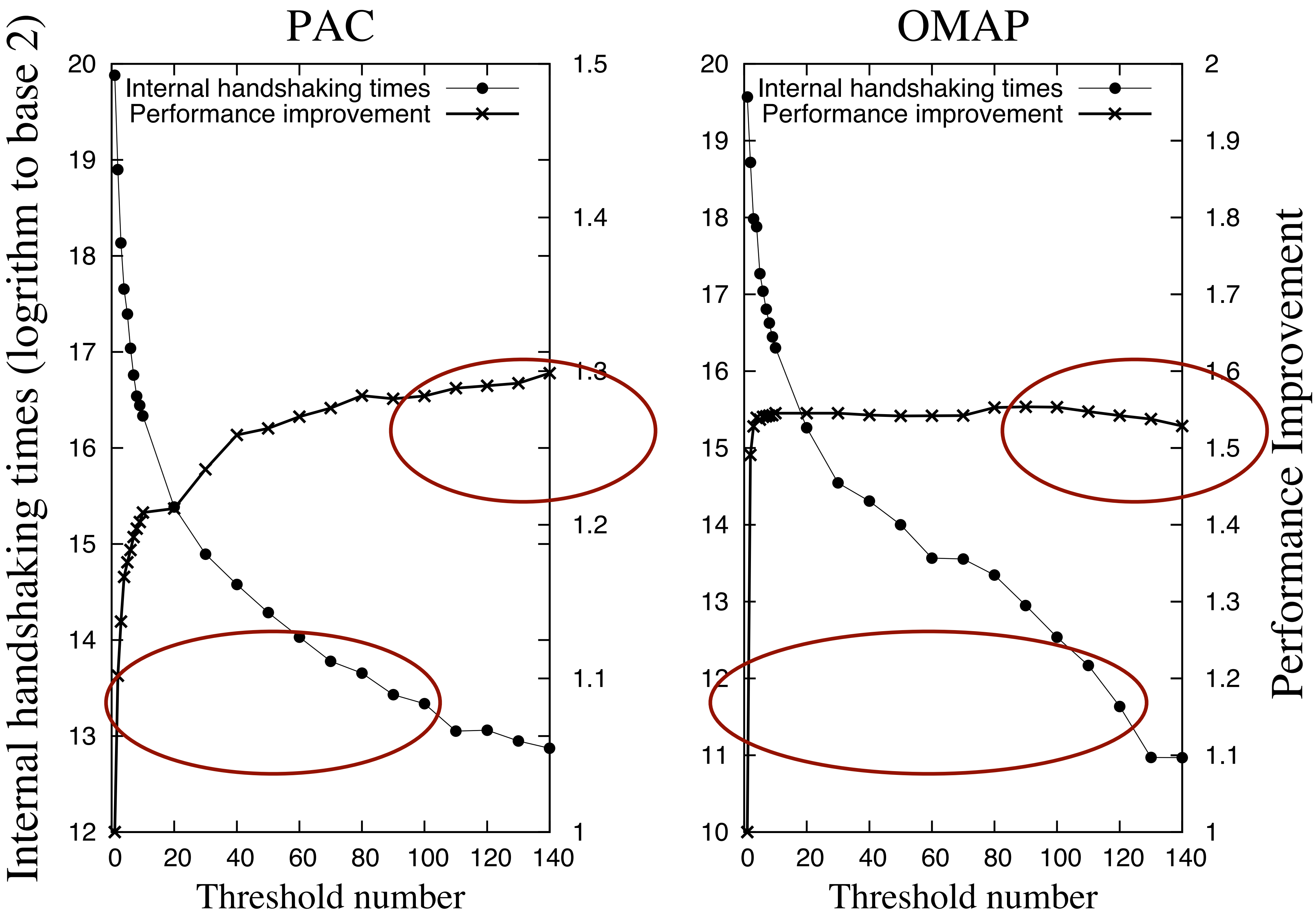
Performance Improvement and Corresponding Internal Handshaking Times: MP3



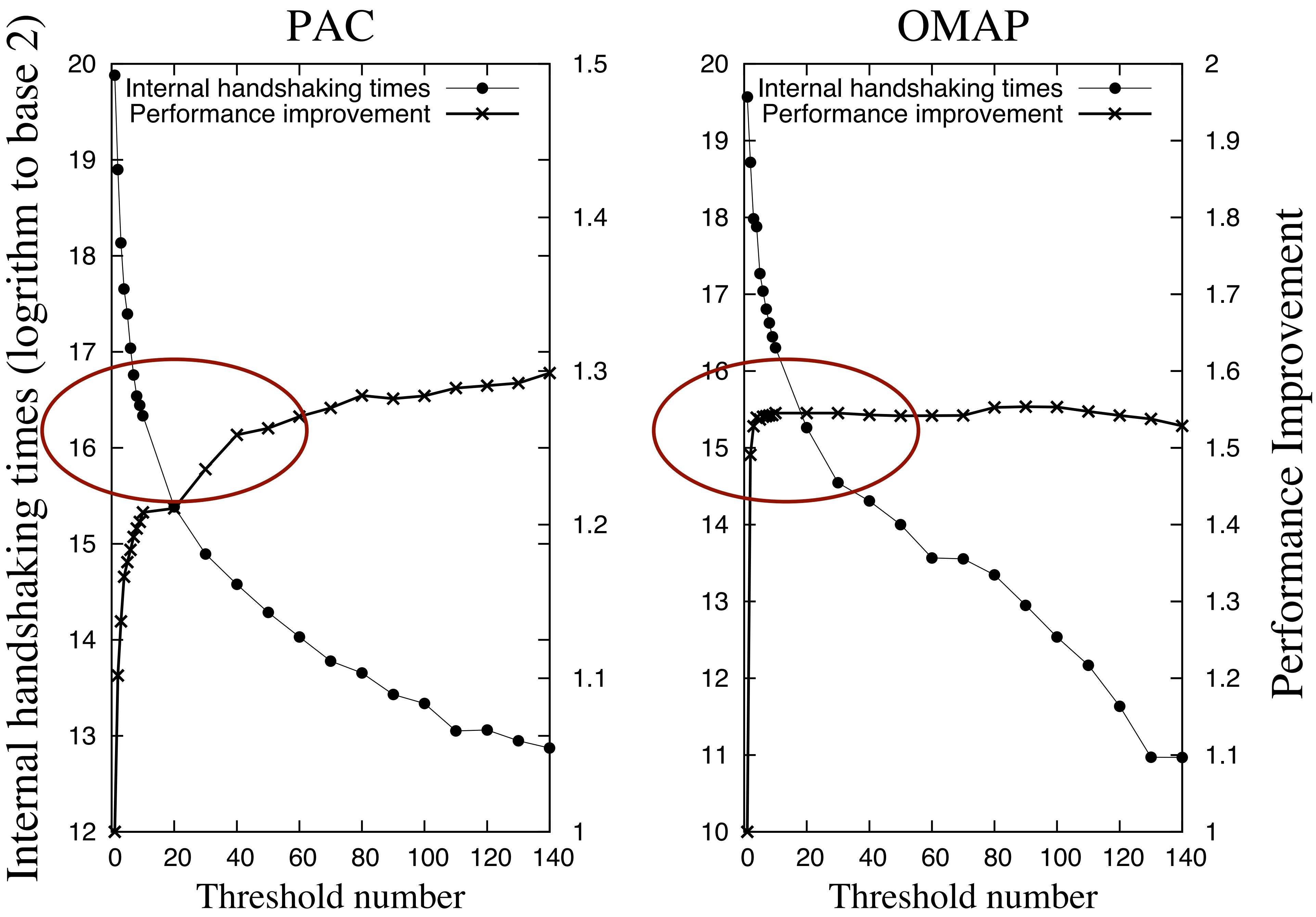
Performance Improvement and Corresponding Internal Handshaking Times: MP3



Performance Improvement and Corresponding Internal Handshaking Times: MP3

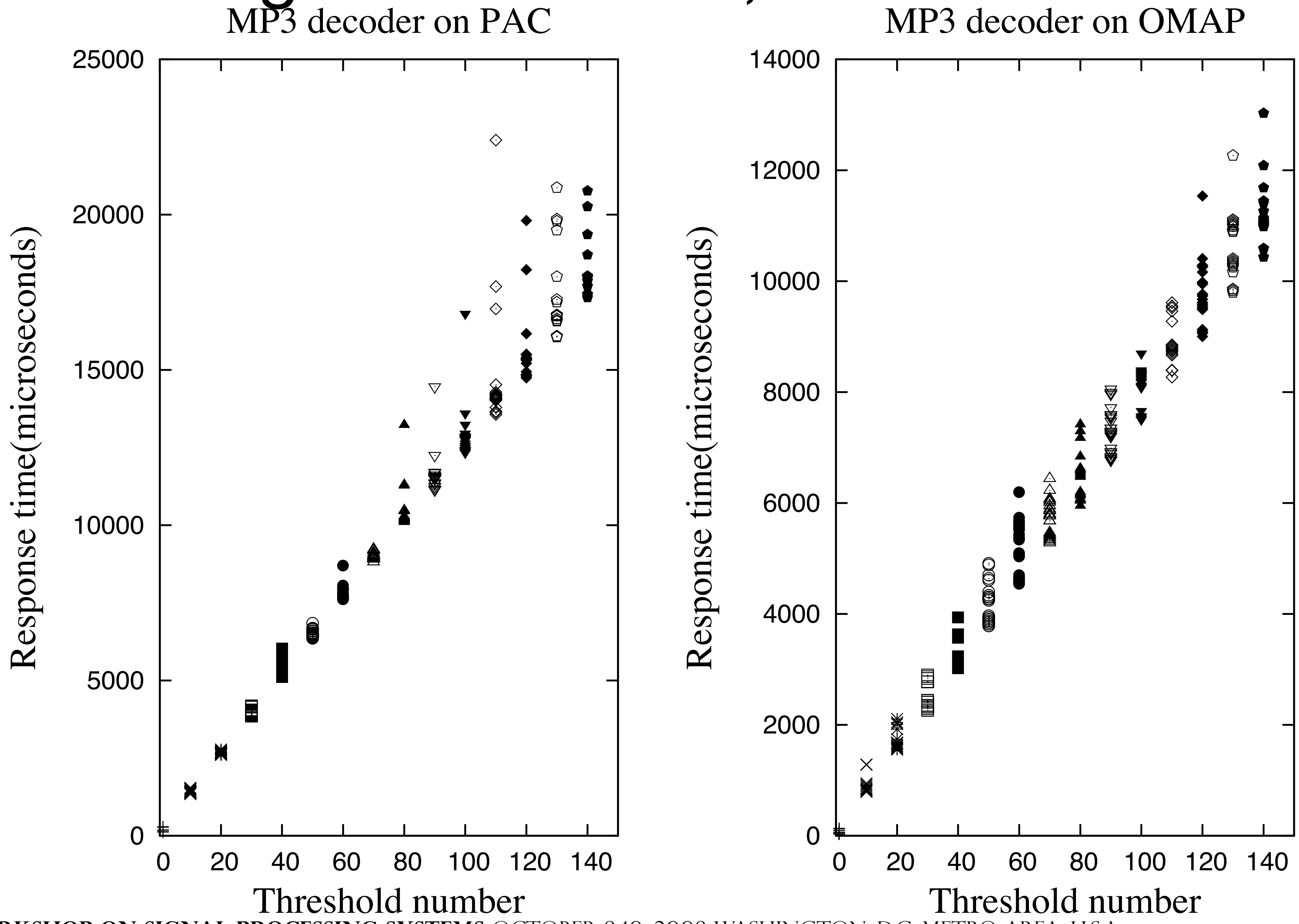


Performance Improvement and Corresponding Internal Handshaking Times: MP3



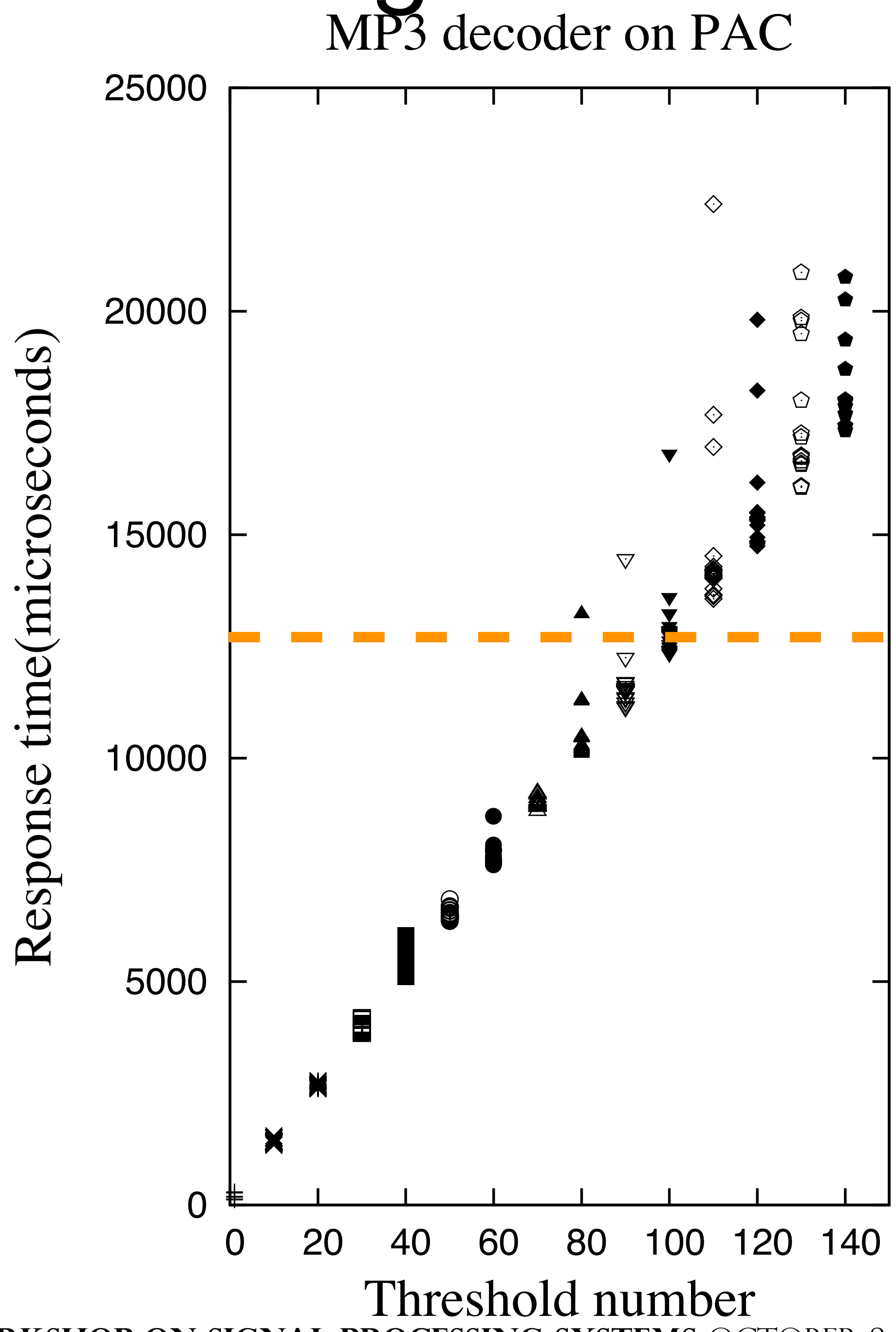
Effect of Threshold to Response Time

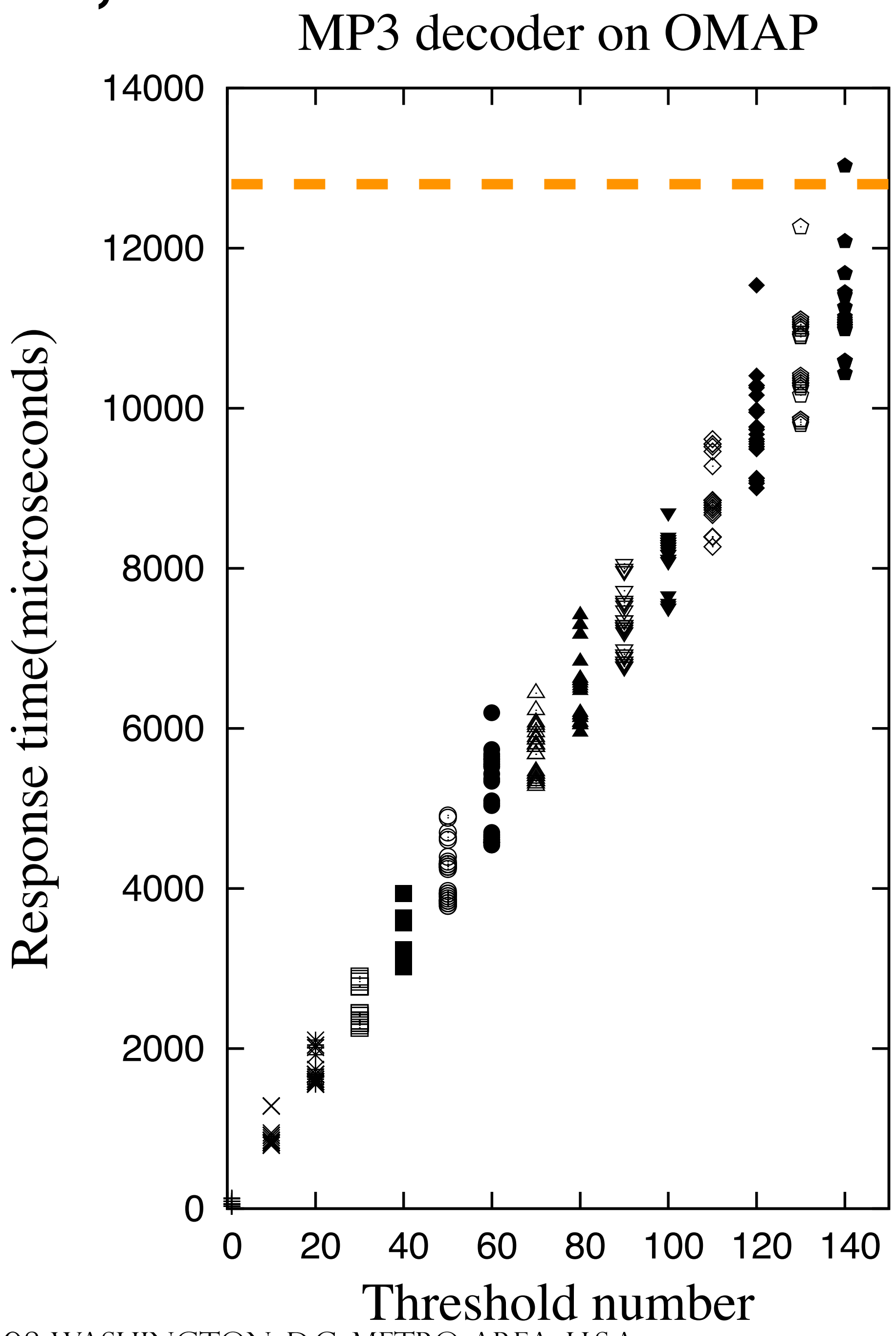
- With timing constraint 12,500 micro-seconds



Effect of Threshold to Response Time

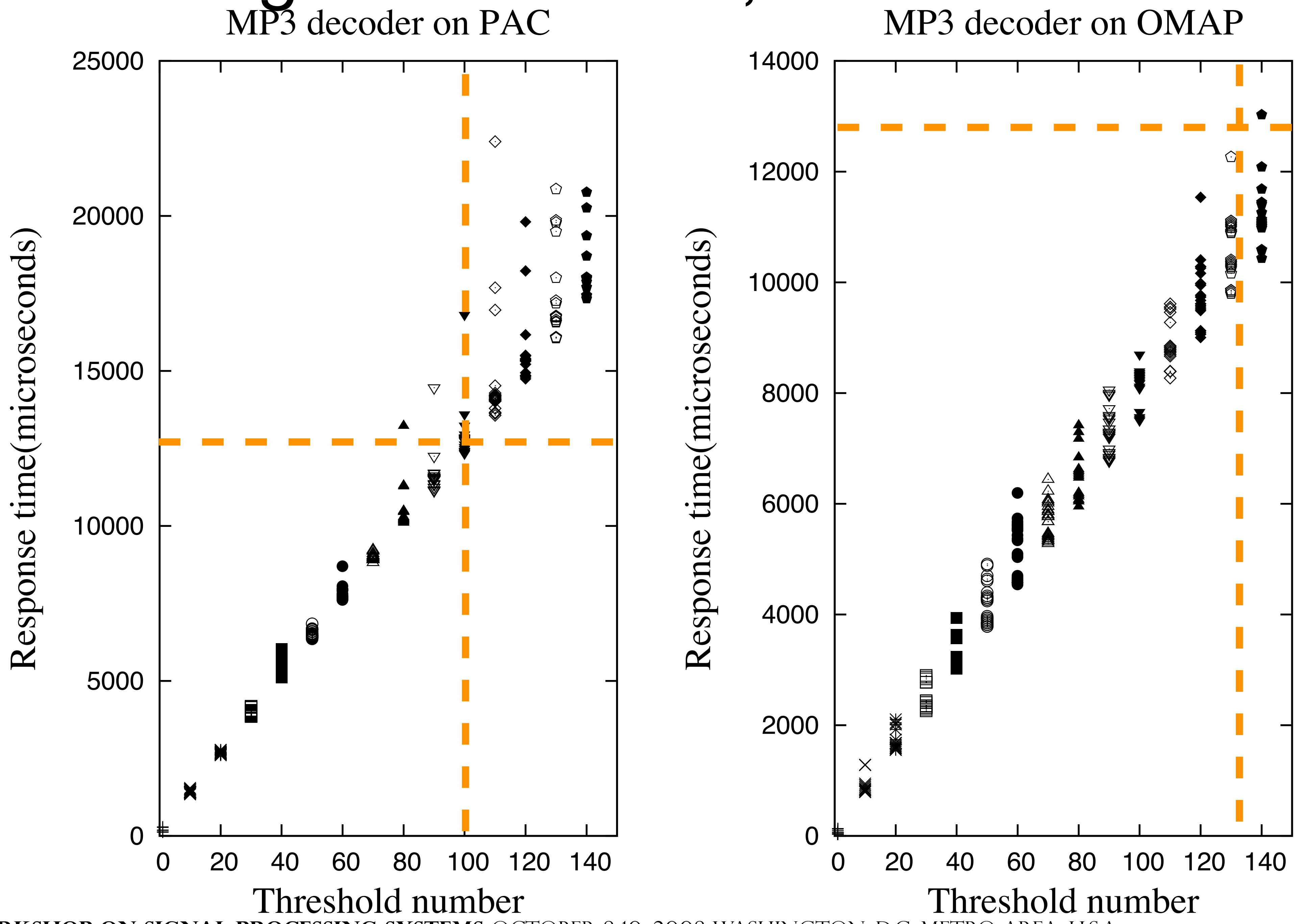
- With timing constraint 12,500 micro-seconds





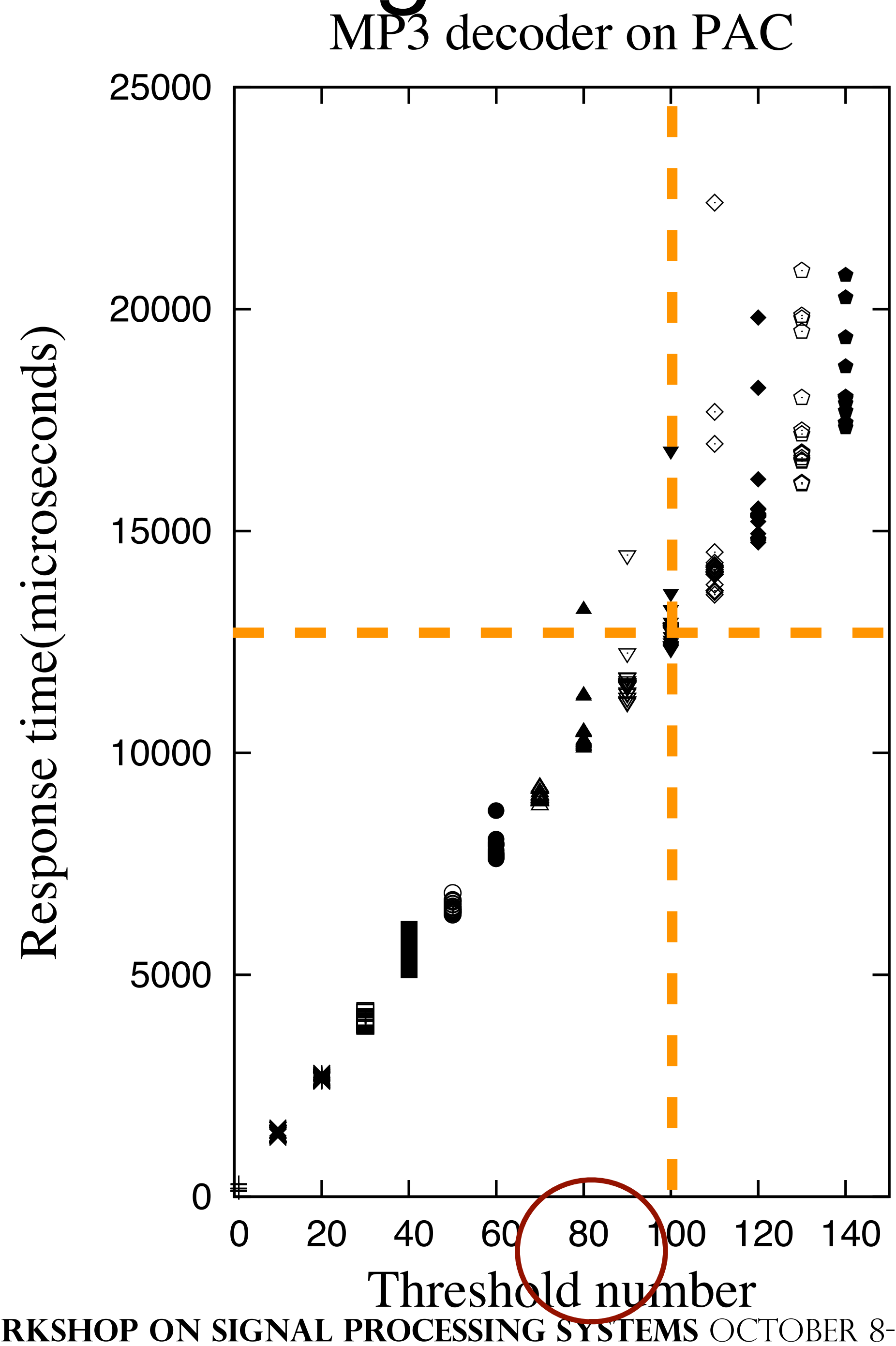
Effect of Threshold to Response Time

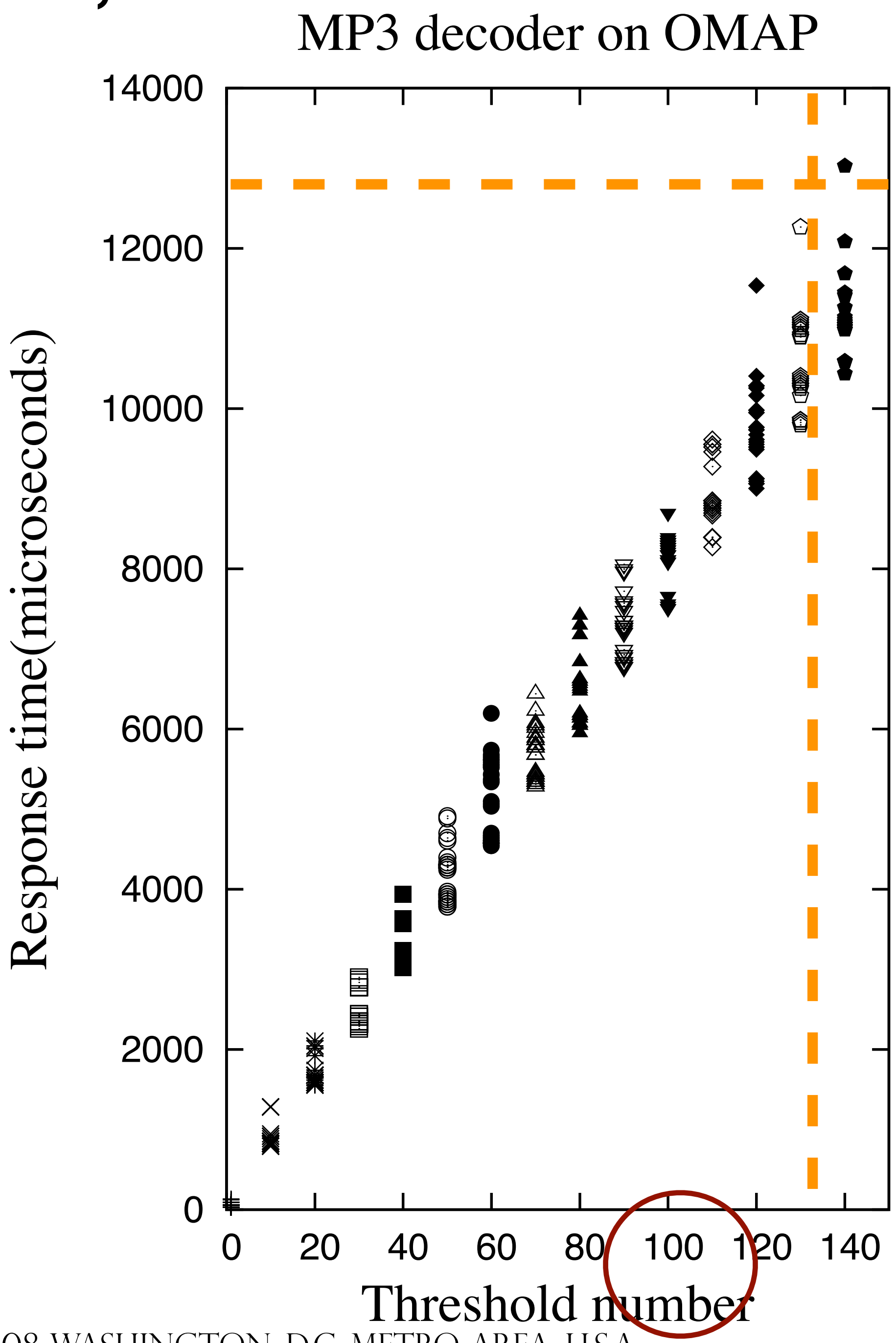
- With timing constraint 12,500 micro-seconds



Effect of Threshold to Response Time

- With timing constraint 12,500 micro-seconds





Summary

- We presented a stream programming model for embedded dual-core processors
- Attempt to provide an abstraction for modeling data streaming applications
  - Communication and computation overlapping
  - Reducing communication overhead
  - Software patterns
- Improve the performance applications
- A methodology of analytic model for reducing internal handshaking times

Thank You!

國立清華大學
National Tsing Hua University