Optimal Multi-Image Processing Streaming Framework on Parallel Heterogeneous Systems

Linh Ha, Jens Krüger, Joao Comba, Sarang Joshi and Cláudio T. Silva

Outline

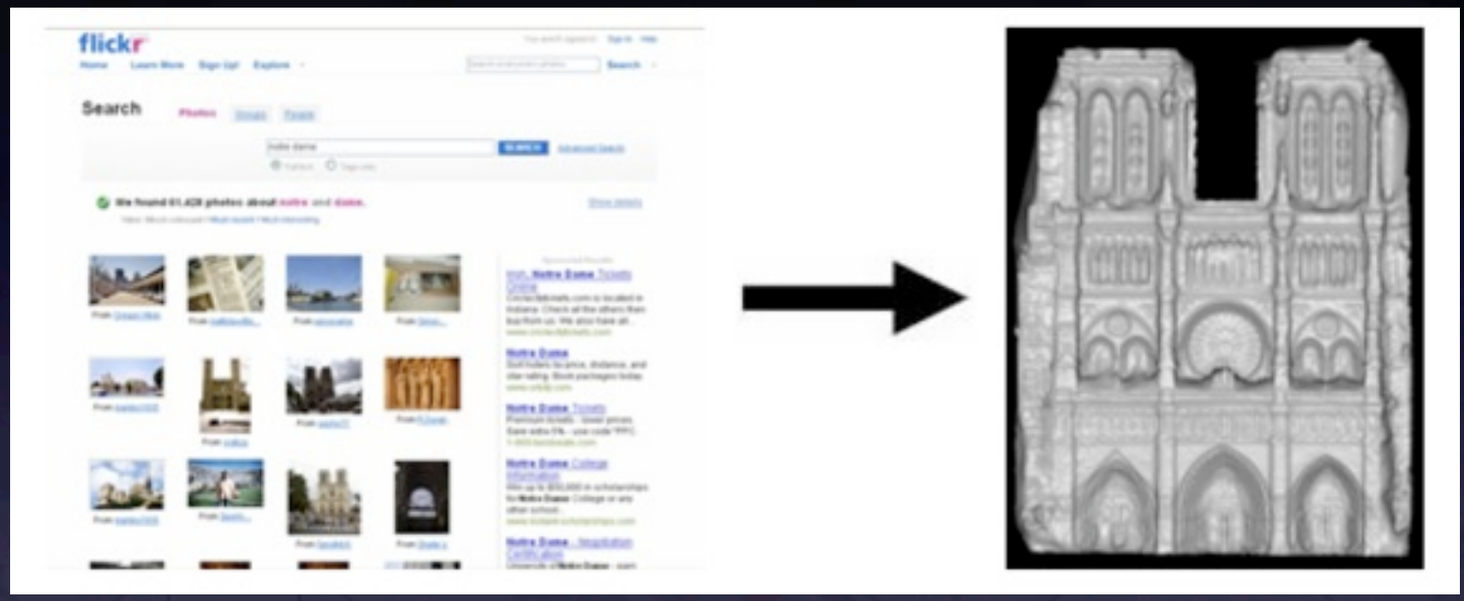
1. Motivation
2. Multi-image processing operators
3. Asynchronous streaming models
4. Degrading problem & solution
5. Conclusion

Outline

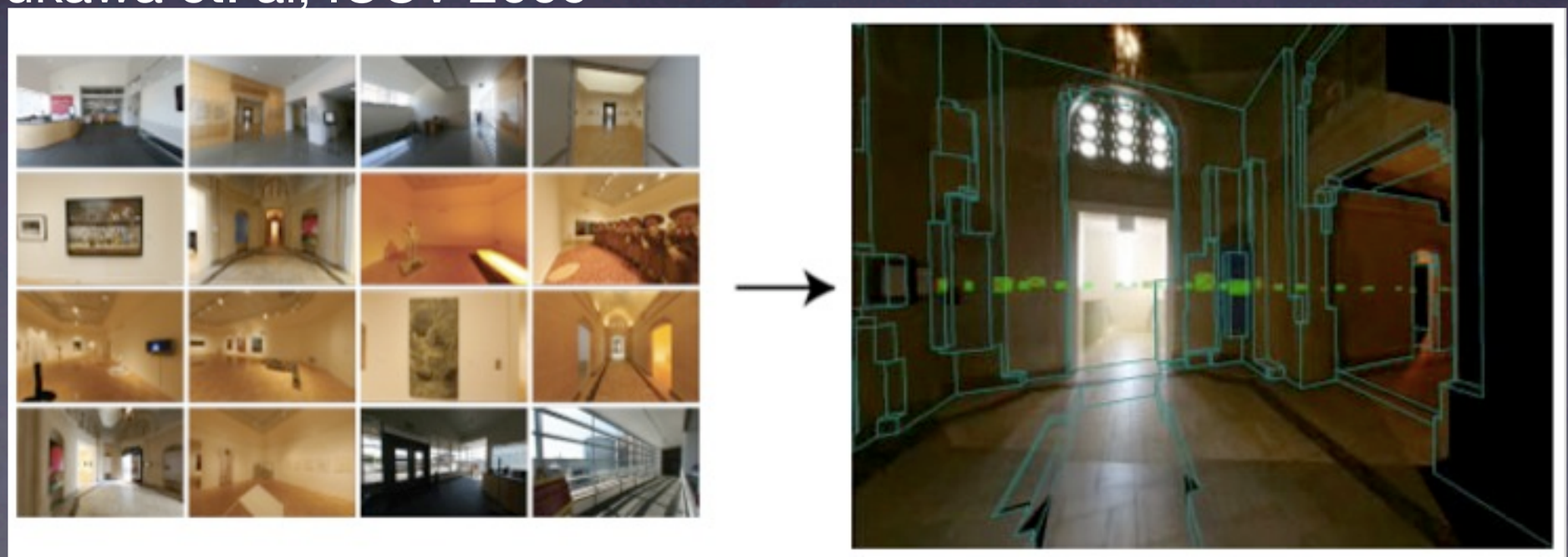
1. Motivation
2. Multi-image processing operators
3. Asynchronous streaming models
4. Degrading problem & solution
5. Conclusion

Motivation

- Multi-View Stereo for Community Photo Collections
  - Goesele et. al, ICCV 2007

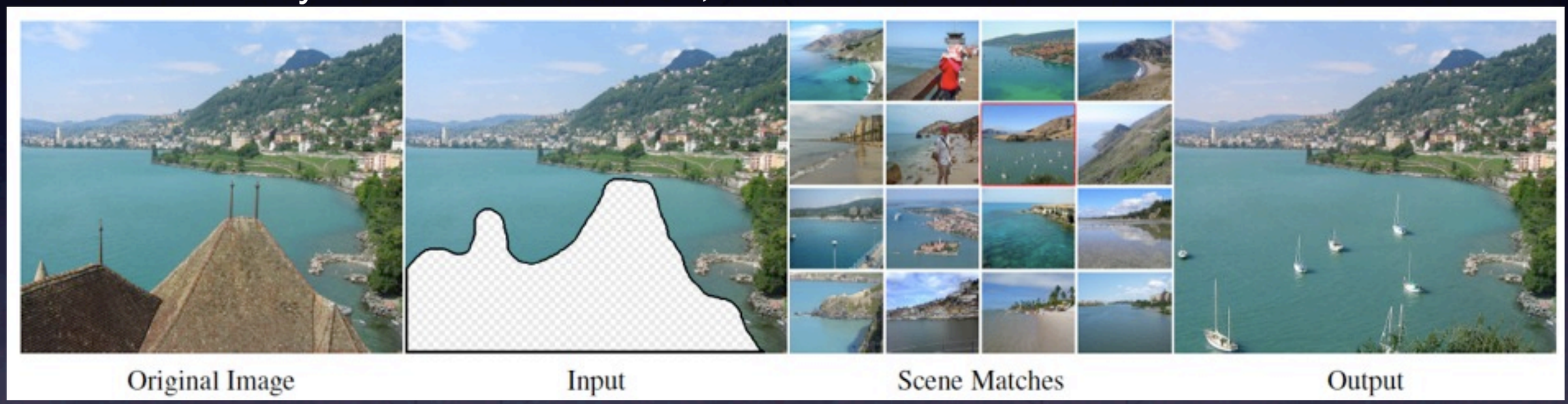


- Reconstructing Building Interiors from Images
  - Furukawa et. al, ICCV 2009

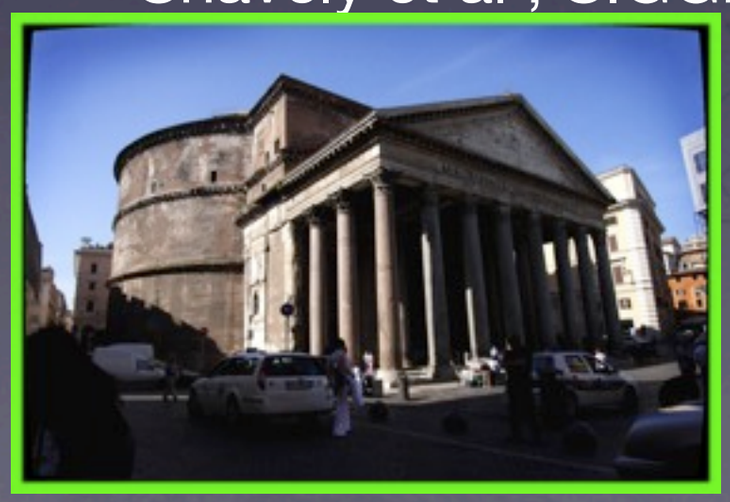


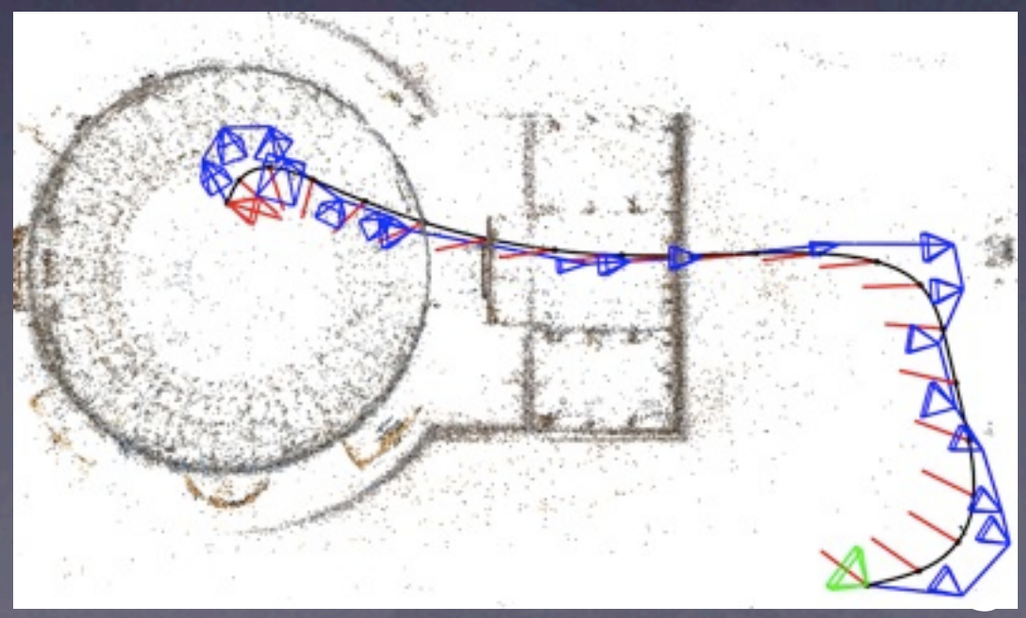
Motivation

- Scene completion using millions of photographs
  - James Hays and Alexei A.Efros, SIGGRAPH 2007



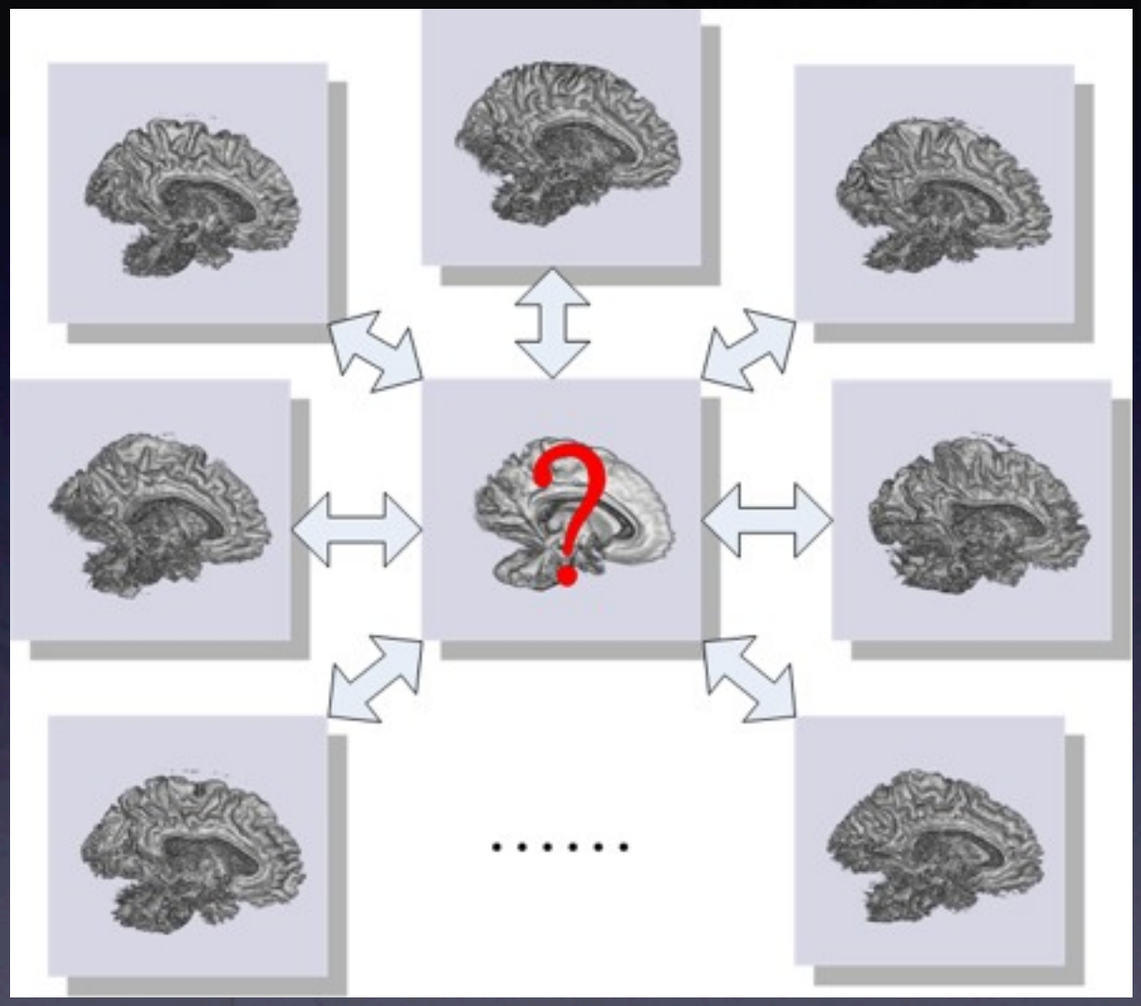
- Finding Paths through the World's Photos
  - Snavely et al , SIGGRAPH 2008





Motivation

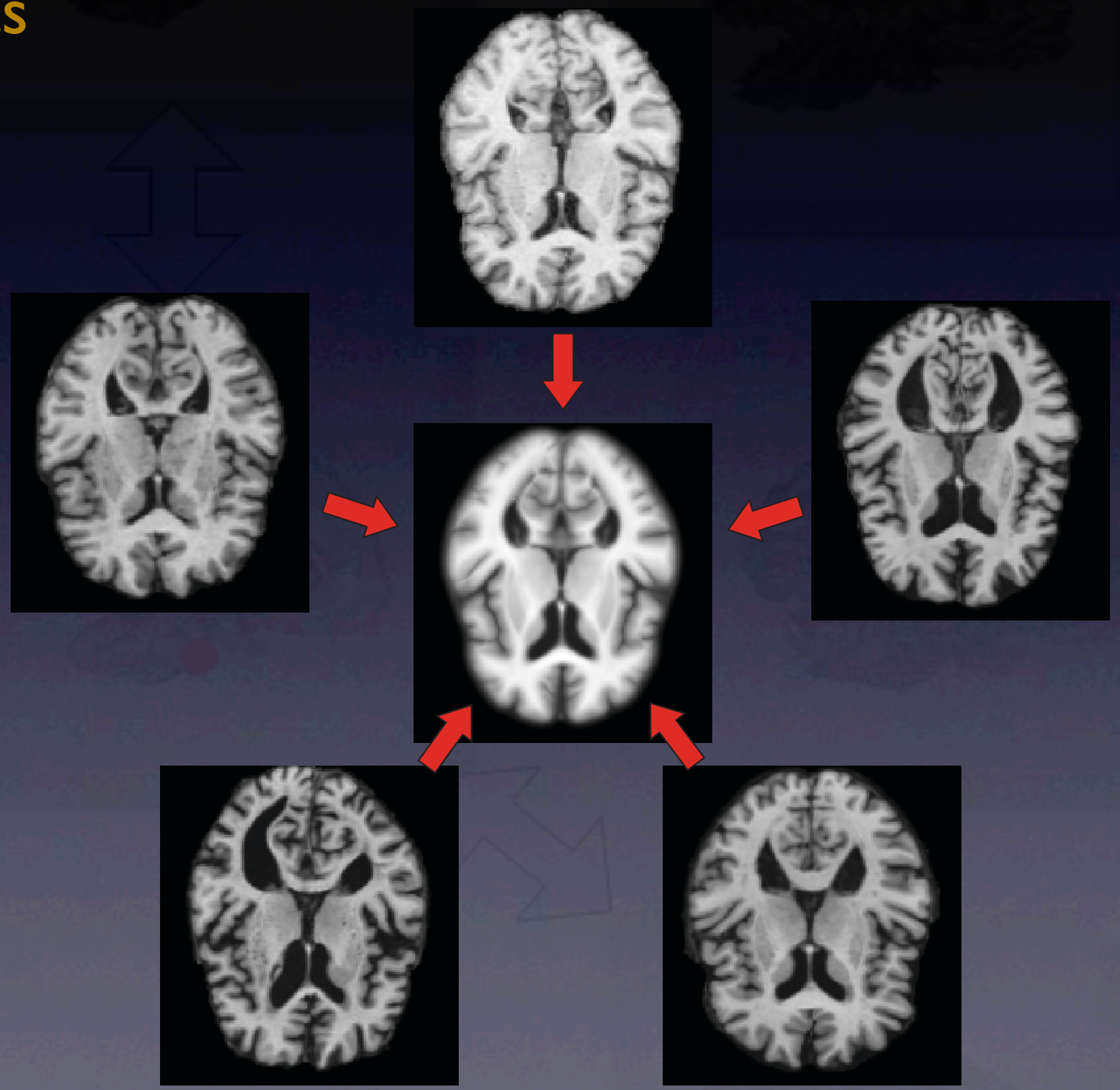
- Atlas construction:
  - Map population into a common coordinate space
    - Learn about variability
    - Describe difference from normal
  - Development, evolution, disease
  - Automatic segmentation



Joshi et al 2004 "Unbiased Diffeomorphic ..."

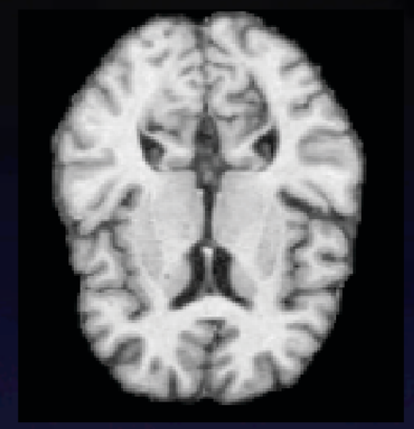
Automatic Segmentation

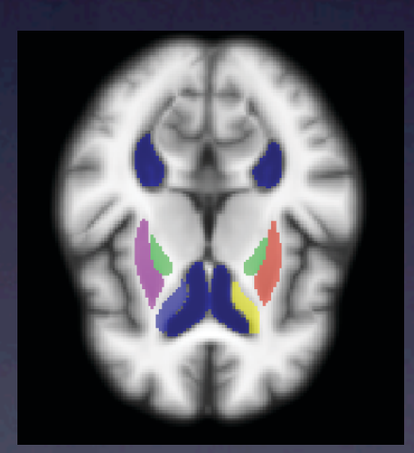
- Compute the atlas

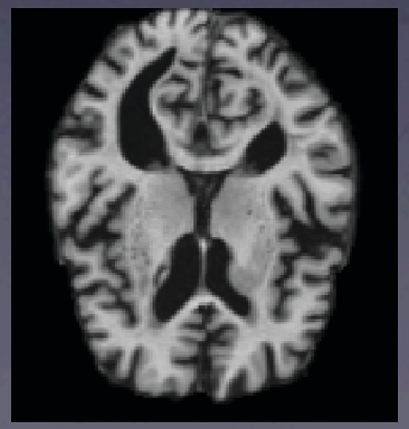


Automatic Segmentation

- Compute the atlas
- Partition the atlas

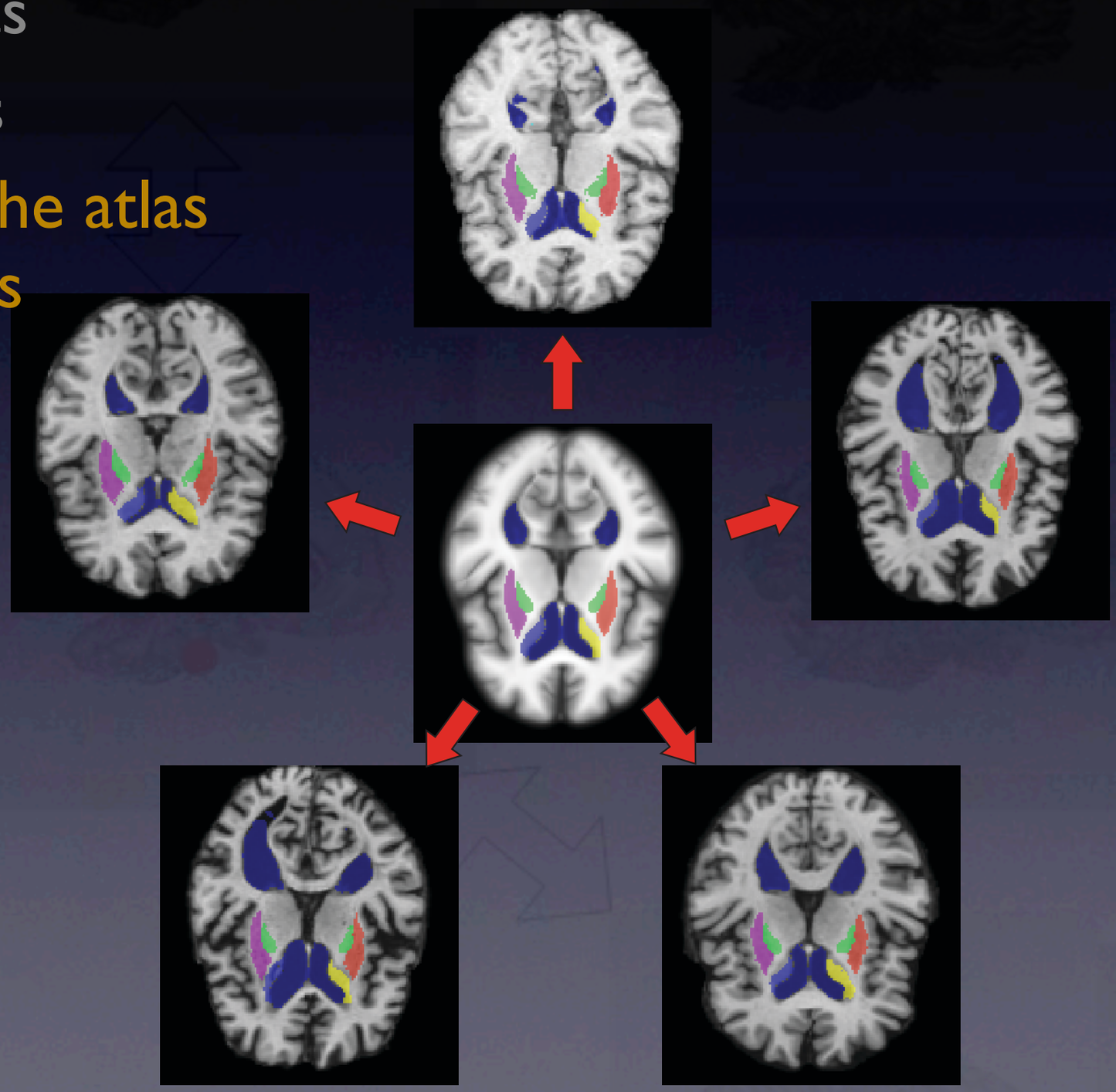






Automatic Segmentation

- Compute the atlas
- Partition the atlas
- Map labels from the atlas back to individuals



Challenges

- Multi-image processing is challenging
  - Large amount of computation
  - Excessive memory requirement
- Solution - parallel processing system:
  - Supercomputer
  - GPU cluster
    - the GPU is proven to be efficient for image processing tasks

However, large scale parallel processing is expensive

Resource and Accessibility trade off

Computing Level	Resources	Accessibility	Applications
GPU Cluster	Abundant	Restricted	Few
Multi-GPU Workstation	Restricted	Feasible	Many
Single GPU desktop	Limited	Widely available	Plenty

Outline

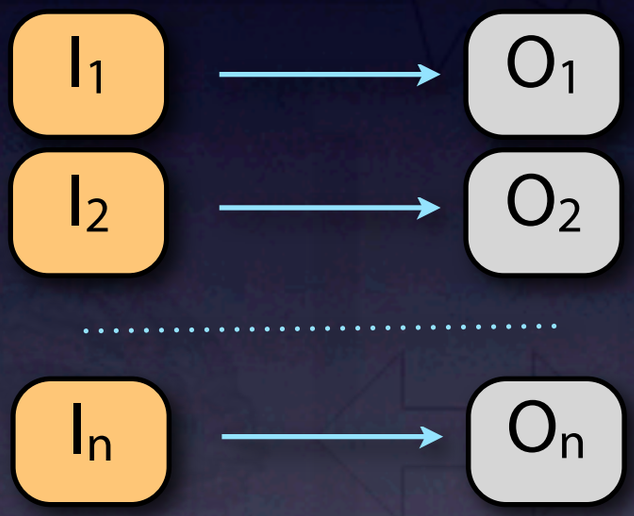
1. Motivation
2. Multi-image processing operators
3. Asynchronous streaming models
4. Degrading problem & solution
5. Conclusion

What are the building blocks of a multi-image processing framework ?

Multi-image Operations

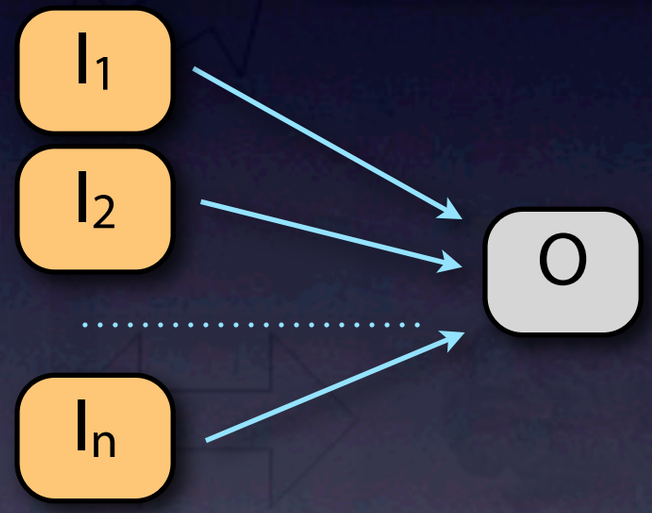
(Flynn's taxonomy classification)

Multi-Input Multi-Output (MIMO)



- add, mul, sub, divide, normalized
- convolution, filter

Multi-Input Single-Output (MISO)

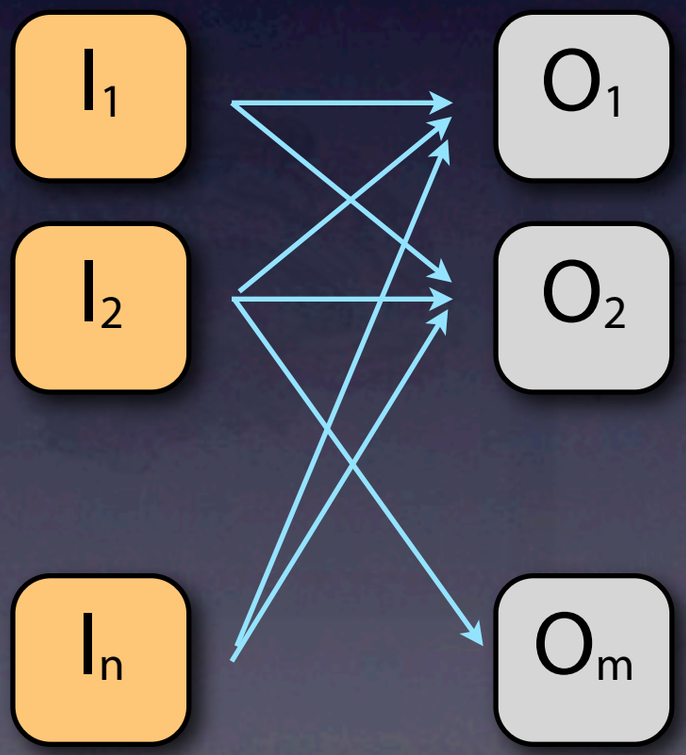


- max, min, range
- average, accumulate

All multi-image operators are presented either by MIMO or MISO

Complex MIMO

Complex
n-inputs & m-outputs

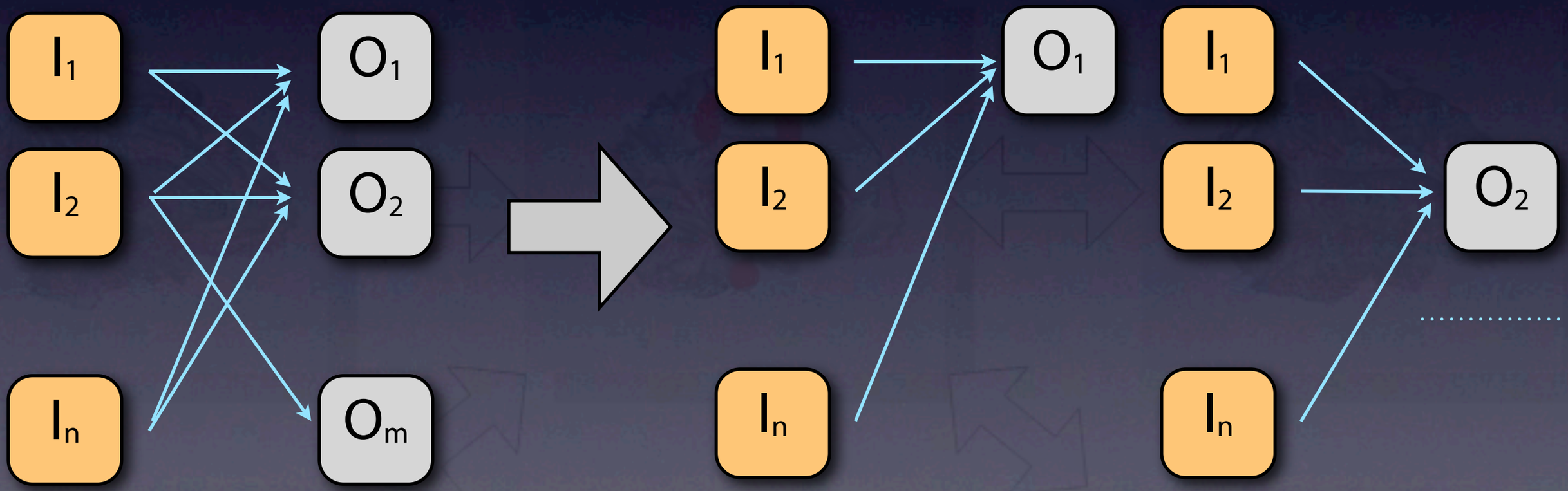


Complex MIMO

Complex
n-inputs & m-outputs

Decomposition

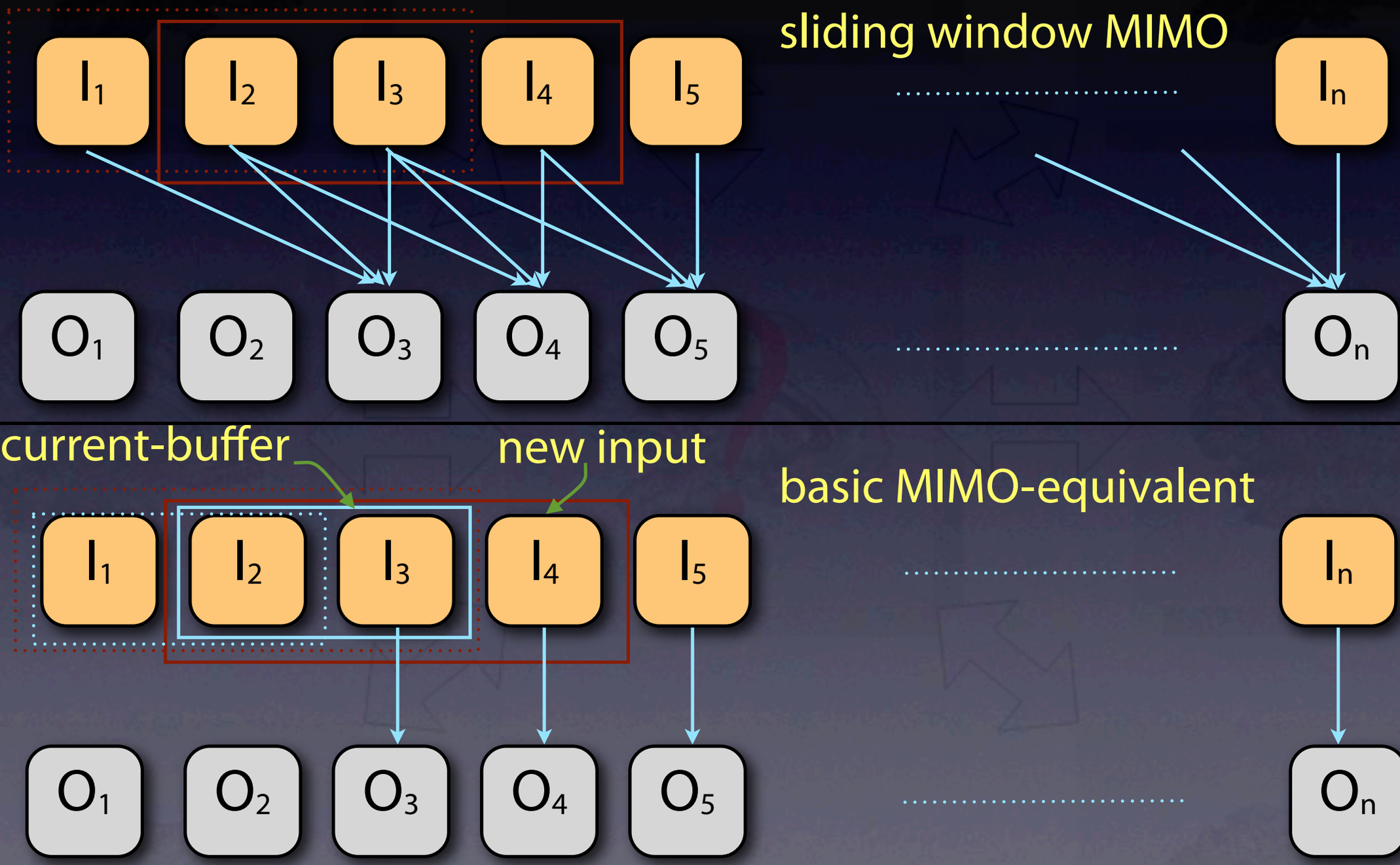
Basic
MIMO & MISO



m instances of MISO

Sliding window MIMO complex

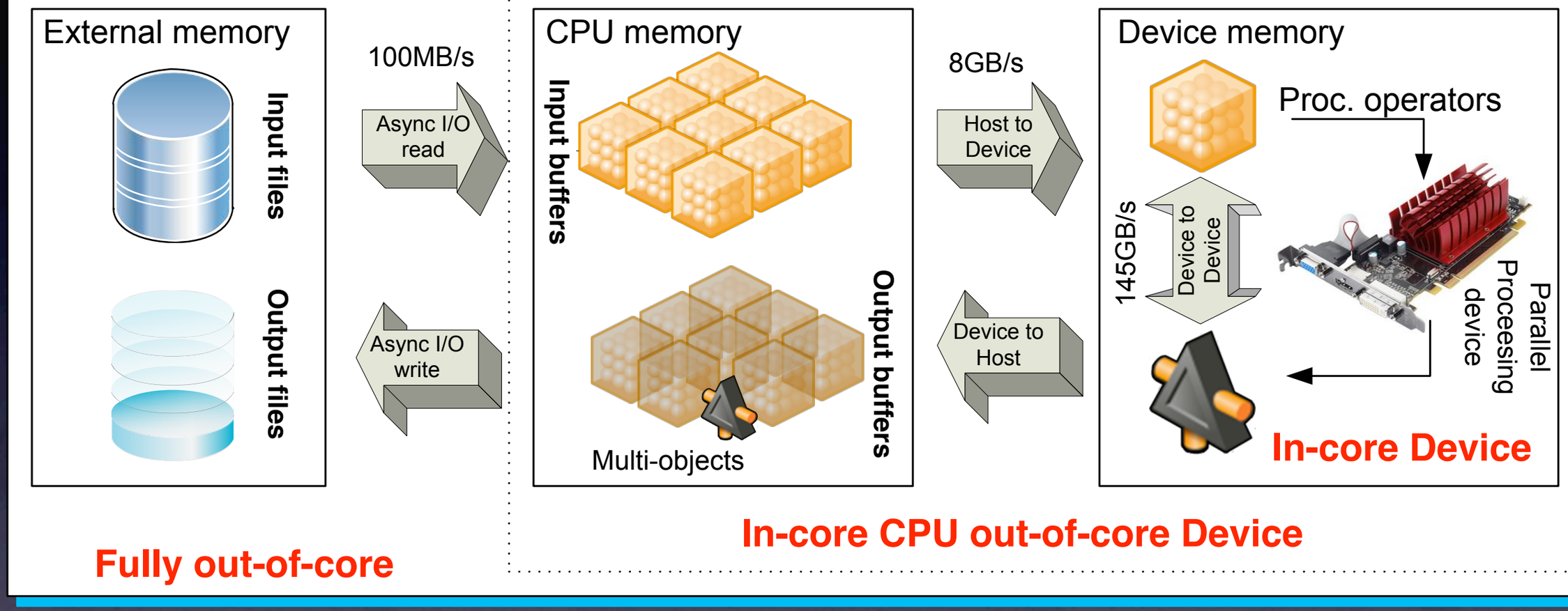
(Functions with memory stages)



Outline

1. Motivation
2. Multi-image processing operators
3. Asynchronous streaming models
4. Degrading problem & solution
5. Conclusion

Out-of-Core MIP Challenges



	External	CPU system memory	GPU global memory
Mem Size	n-TB	10-256GB	512MB-6GB
Bandwidth	100MB/s	3-8 GB/s	140-200GB/s

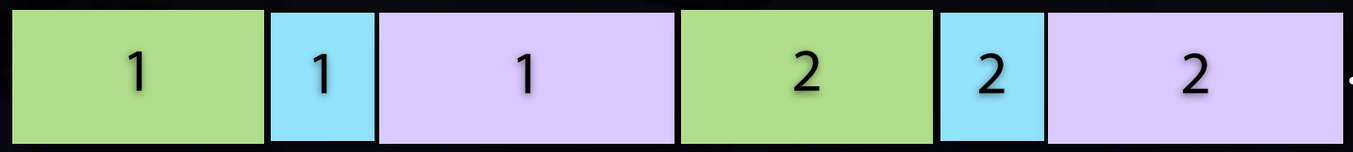
1-mag difference

2-mags difference

Data transfer becomes bottleneck ! Solution: Asynchronous processing

Analysis constraints

MI operator



- Order independent processing
  - We can order the data and perform the processing in that order

Satisfied with functions in practice

- Performance assumption
  - Computation are uniform
  - Data transfer are uniform

Mostly satisfied with preprocessed data

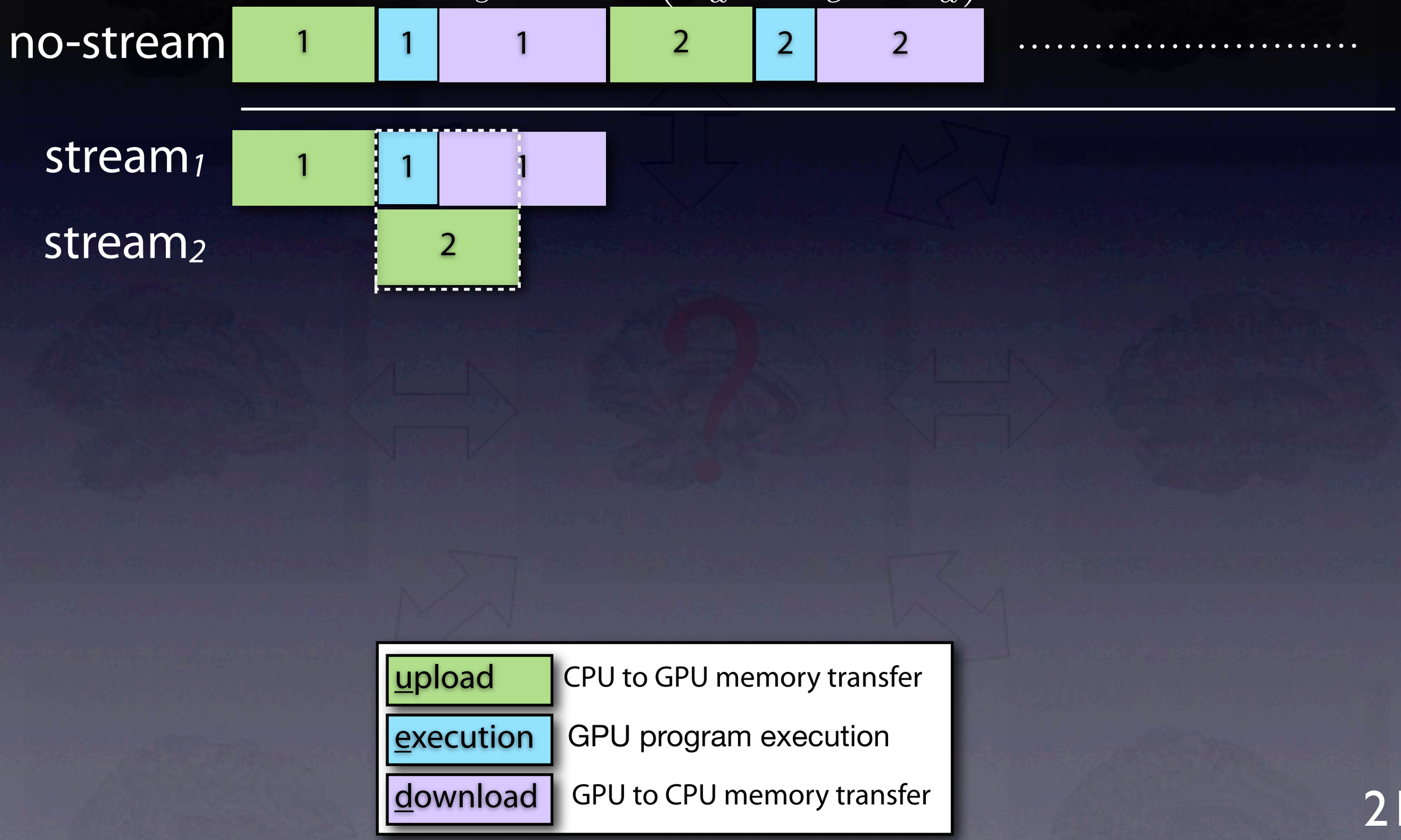
- Simple 3-stage pipeline

upload	CPU to GPU memory transfer
execution	GPU program execution
download	GPU to CPU memory transfer

First level of out-of-core processing models

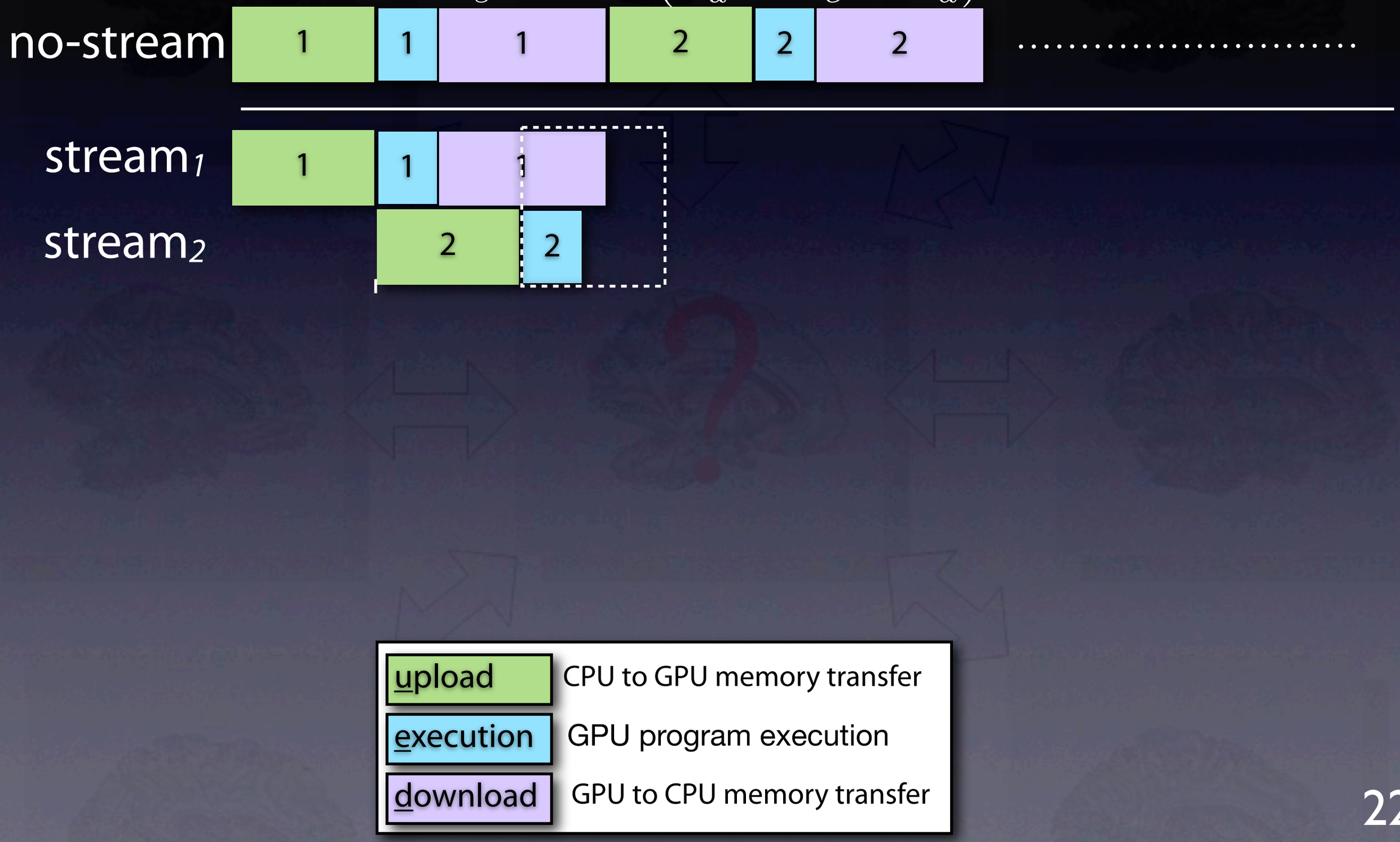
Asynchronous versus Synchronous

$$T_s = n \times (T_u + T_e + T_d)$$



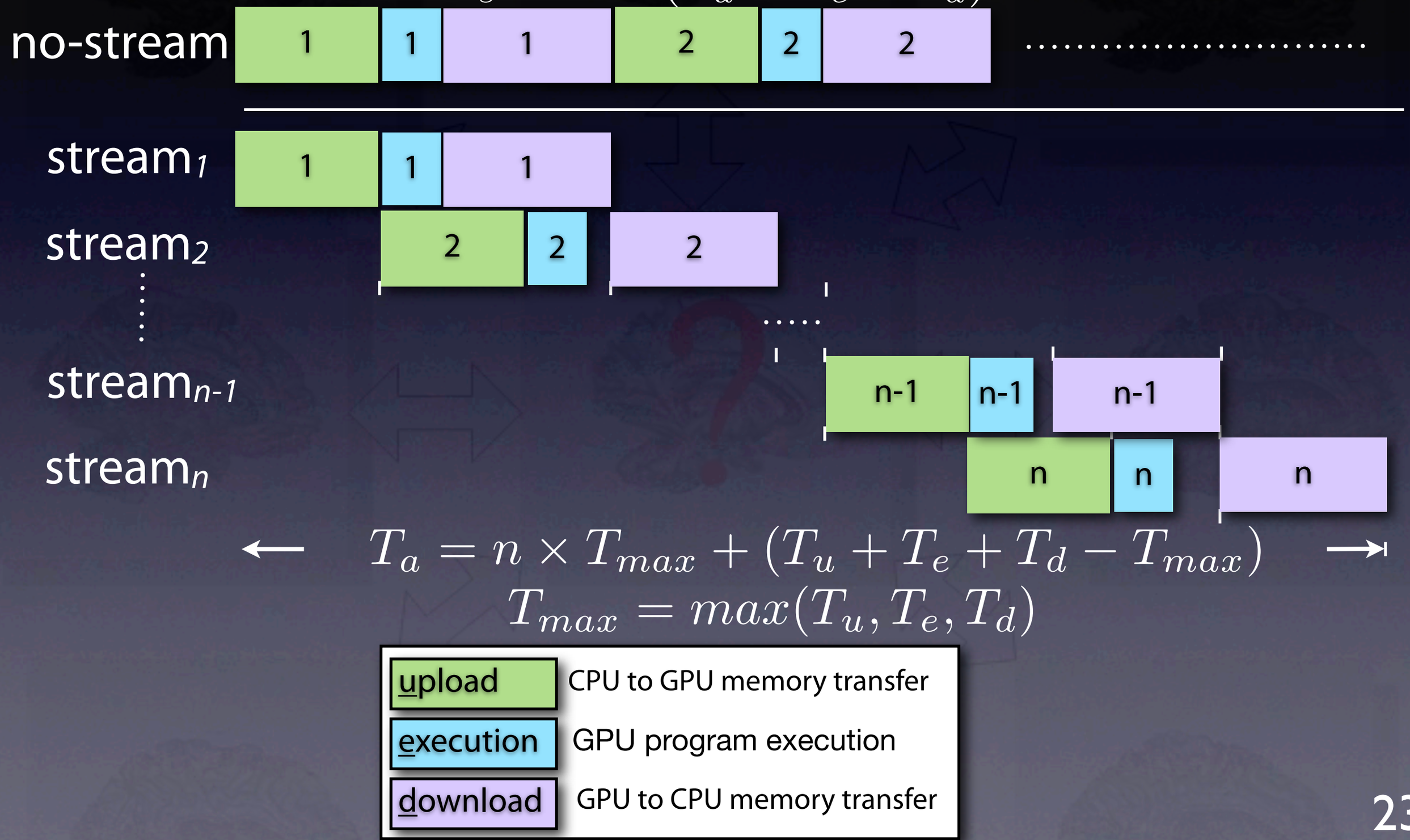
Asynchronous versus Synchronous

$$T_s = n \times (T_u + T_e + T_d)$$



Asynchronous versus Synchronous

$$T_s = n \times (T_u + T_e + T_d)$$



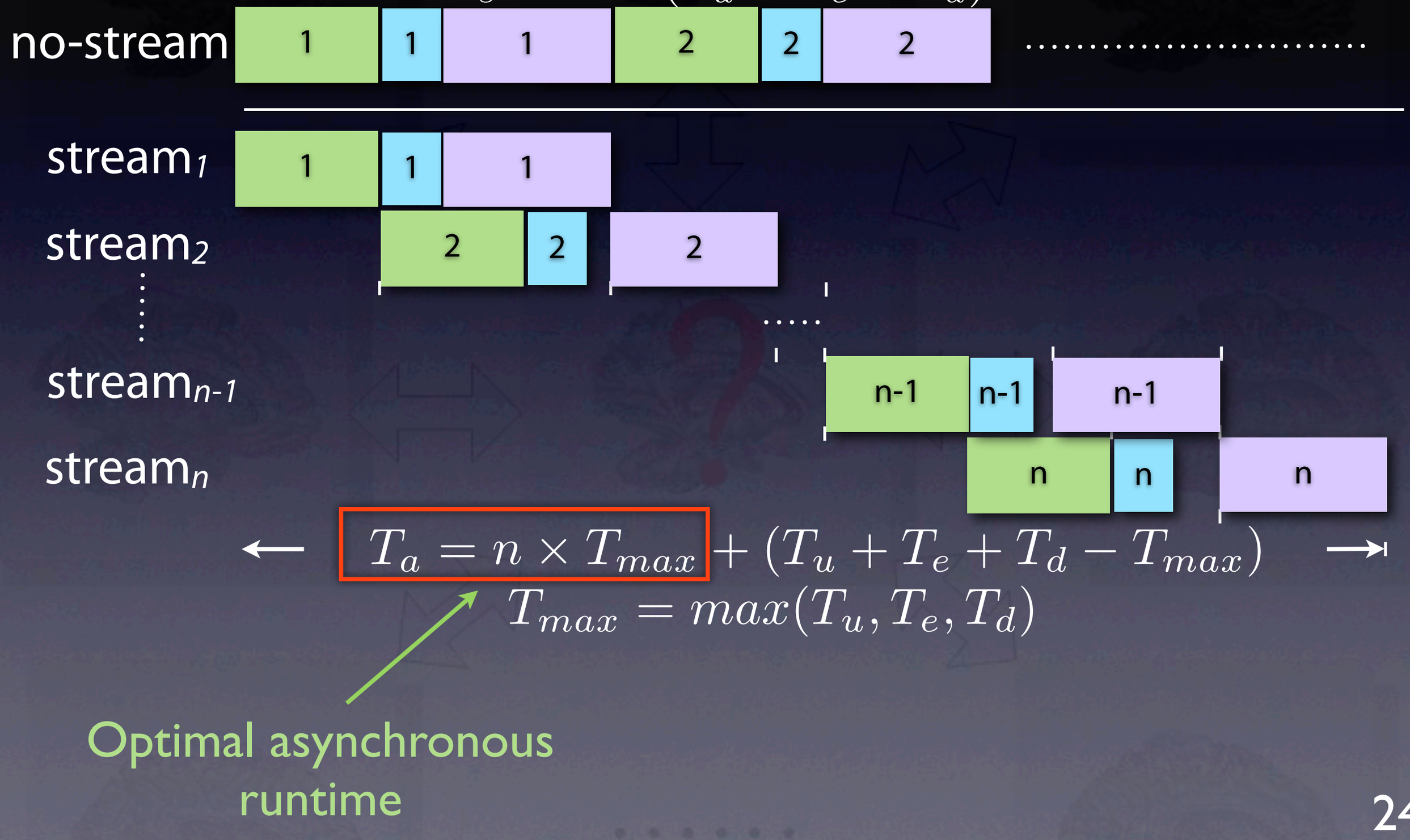
$$\leftarrow \quad T_a = n \times T_{max} + (T_u + T_e + T_d - T_{max}) \quad \rightarrow$$

$$T_{max} = max(T_u, T_e, T_d)$$

upload	CPU to GPU memory transfer
execution	GPU program execution
download	GPU to CPU memory transfer

Asynchronous versus Synchronous

$$T_s = n \times (T_u + T_e + T_d)$$



$$\leftarrow \quad \boxed{T_a = n \times T_{max}} + (T_u + T_e + T_d - T_{max}) \quad \rightarrow$$

$$T_{max} = max(T_u, T_e, T_d)$$

Optimal asynchronous runtime

Asynchronous versus Synchronous

$n = 10$

$$T_s = n \times (T_u + T_e + T_d) = 1300$$

$$\leftarrow \quad T_a = n \times T_{max} + (T_u + T_e + T_d - T_{max}) = 670 \rightarrow$$

$$T_{max} = max(T_u, T_e, T_d) = 60$$

Asynchronous versus Synchronous

Implicit streaming model

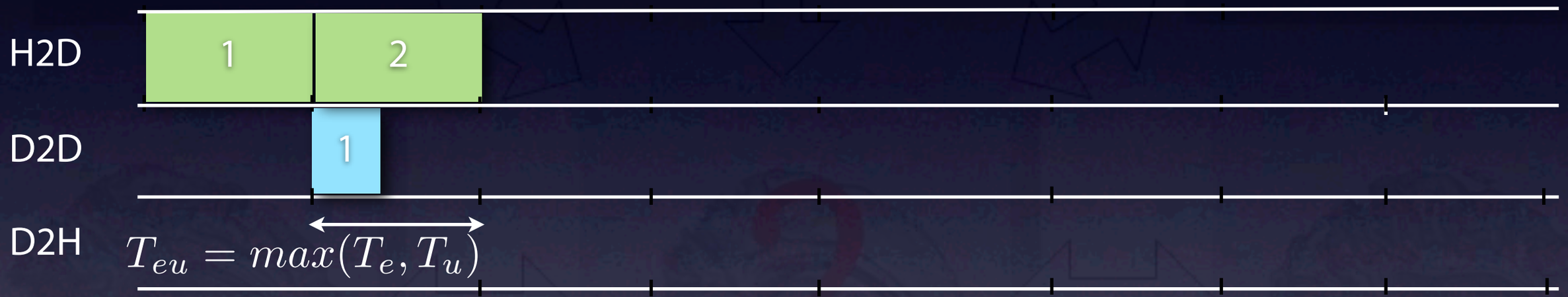
- Logical model
- Simple but cannot be used for out-of-core

Explicit streaming model

- Hardware aware
- Hardware independent

Hardware-aware Execution

Stream is assigned after the hardware execution unit



upload	CPU to GPU memory transfer
execution	GPU program execution
download	GPU to CPU memory transfer

The system with two data transfer + one execution units

Hardware-aware Execution

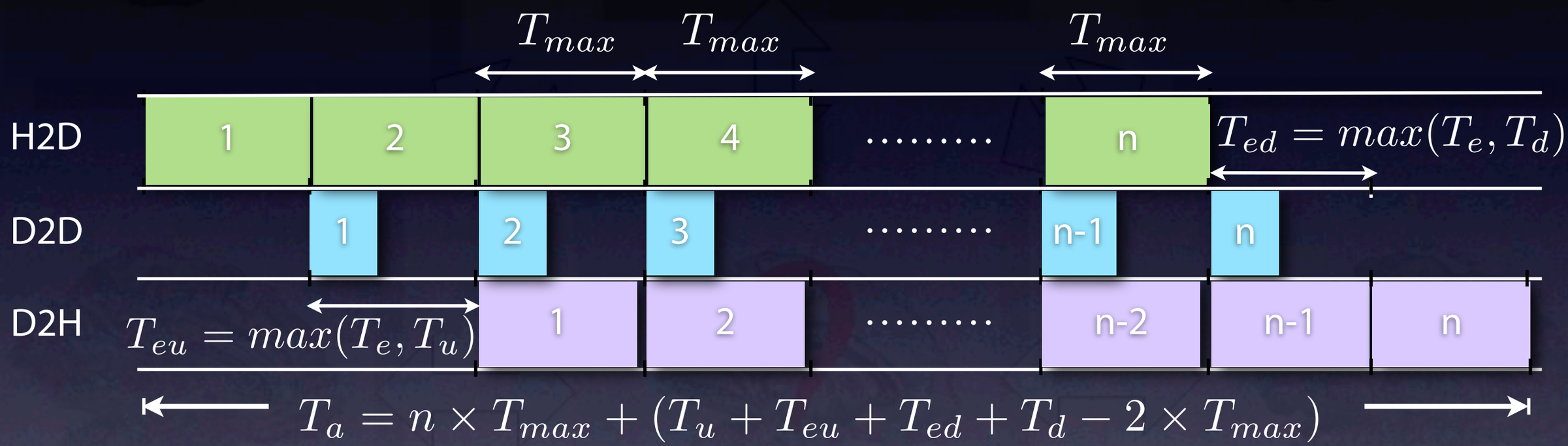
Stream is assigned after the hardware execution unit

upload	CPU to GPU memory transfer
execution	GPU program execution
download	GPU to CPU memory transfer

The system with two data transfer + one execution units

Hardware-aware Execution

Stream is assigned after the hardware execution unit

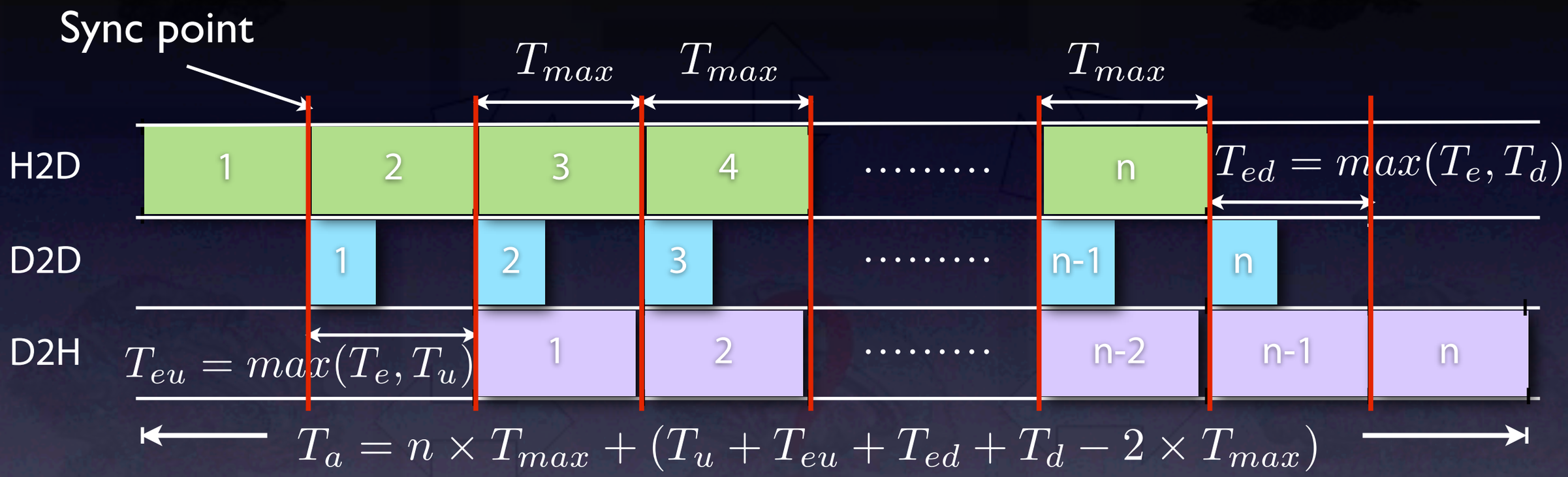


upload	CPU to GPU memory transfer
execution	GPU program execution
download	GPU to CPU memory transfer

The system with two data transfer + one execution units

Hardware-aware Execution

Stream is assigned after the hardware execution unit

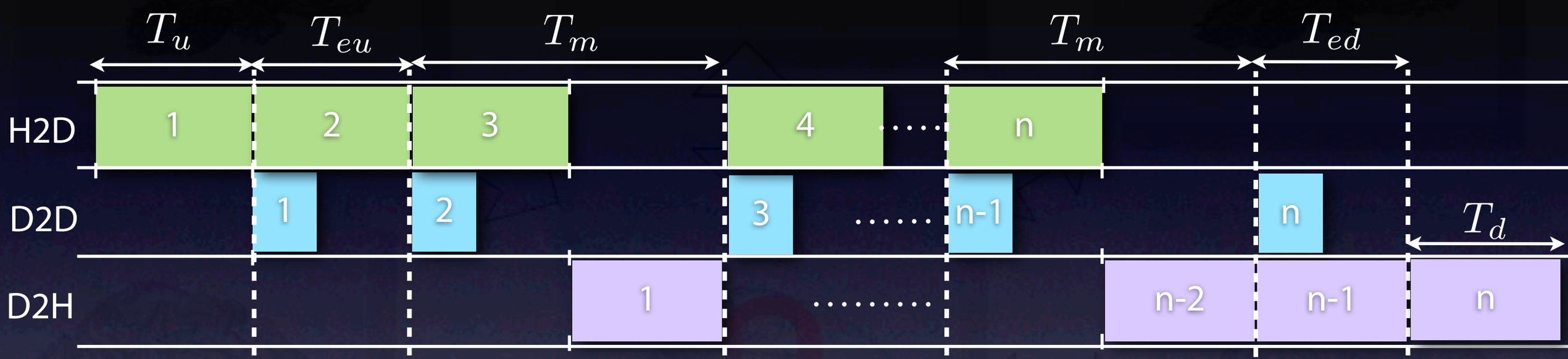


Hardware aware execution

- Physical model - hardware dependent model
- Memory consumption = number of execution units
- Maximal speed up factor = number of execution units

Hardware-independent Execution

Stream is assigned after the function stage



$$T_a = T_u + T_{eu} + (n-2) \times T_m + T_{ed} + T_d$$
$$T_m = max(T_u + T_d, T_e)$$

The system with one data transfer + one execution units

Hardware-independent Execution

Stream is assigned after the function stage

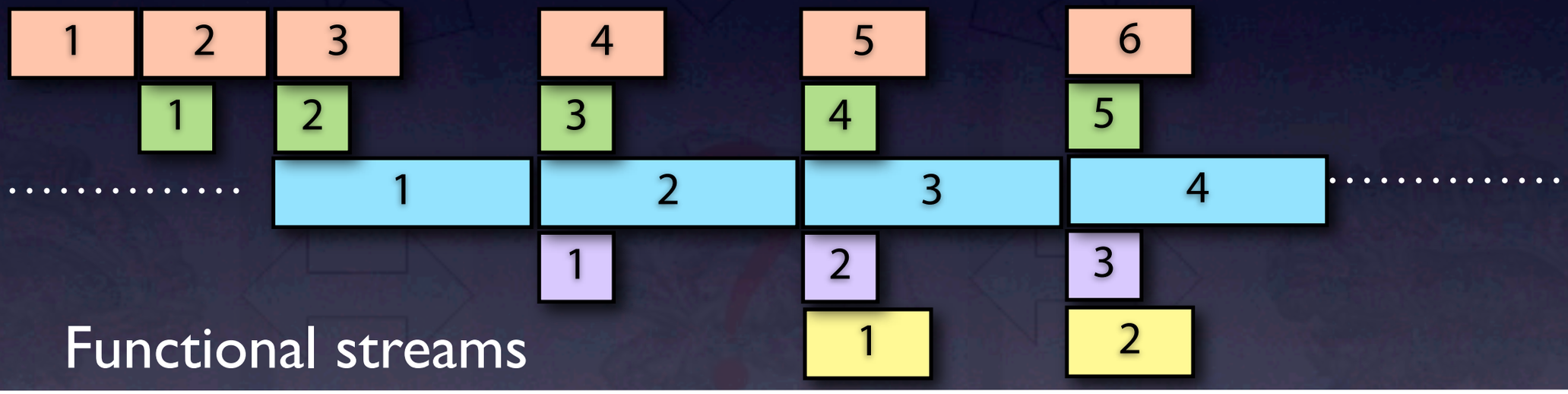
$$T_a = T_u + T_{eu} + (n-2) \times T_m + T_{ed} + T_d$$
$$T_m = max(T_u + T_d, T_e)$$

Hardware independent execution

- Functional model - hardware independent model
- Memory consumption = number of streams
- Potential speed up factor of the algorithm
  = number of streams

Extension to Full Out-of-Core MIP

Full Out-Of-Core = Out-Of-Core GPU In-core CPU + Out-Of-Core CPU transfers

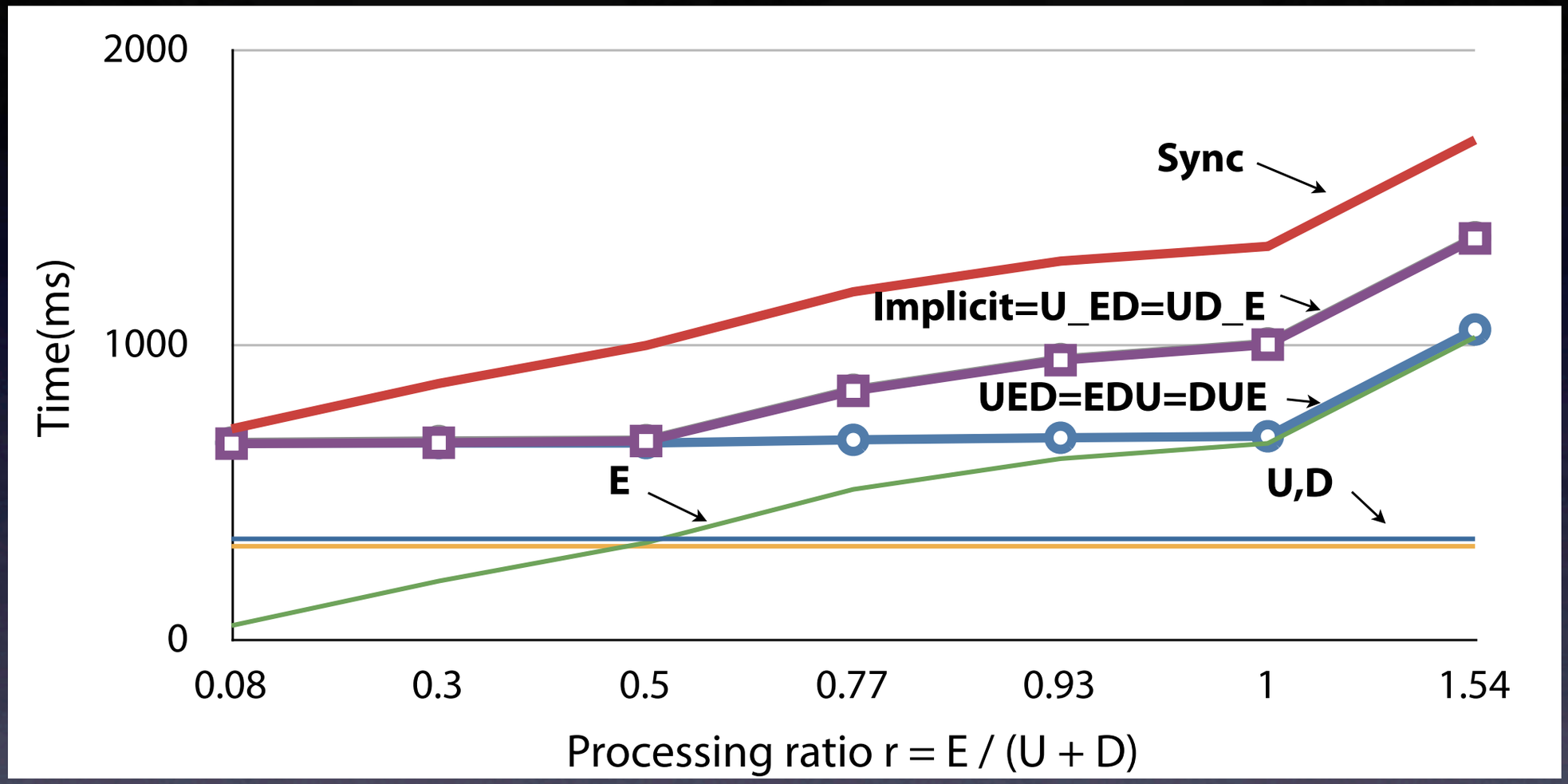


Functional streams

upload	CPU to GPU memory transfer	d-upload	disk to CPU memory transfer
execution	GPU program execution	d-download	CPU to disk memory transfer
download	GPU to CPU memory transfer		

The hardware independent extension to full out-of-core MIP is proven to be optimal

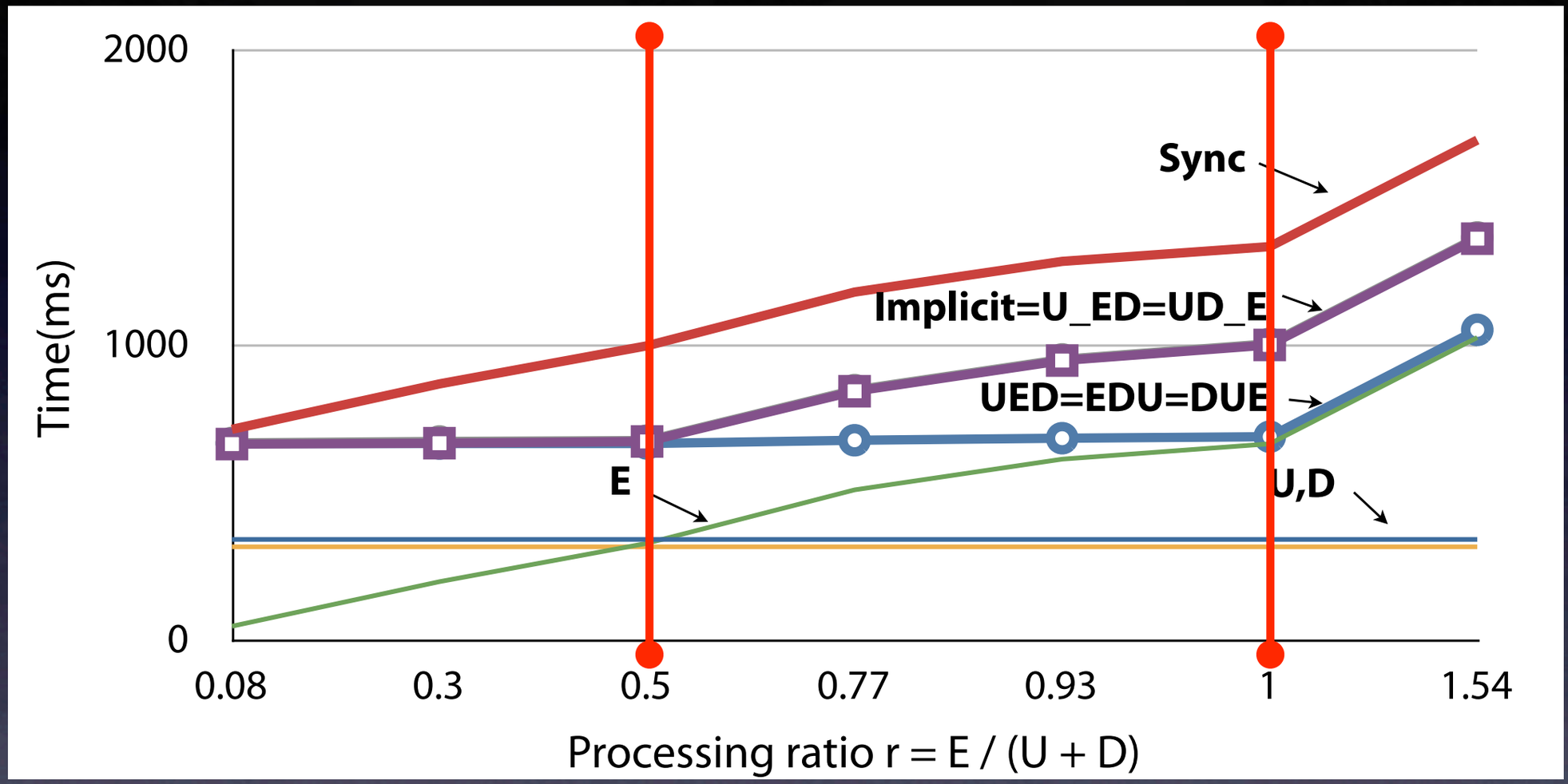
Ideal condition runtime

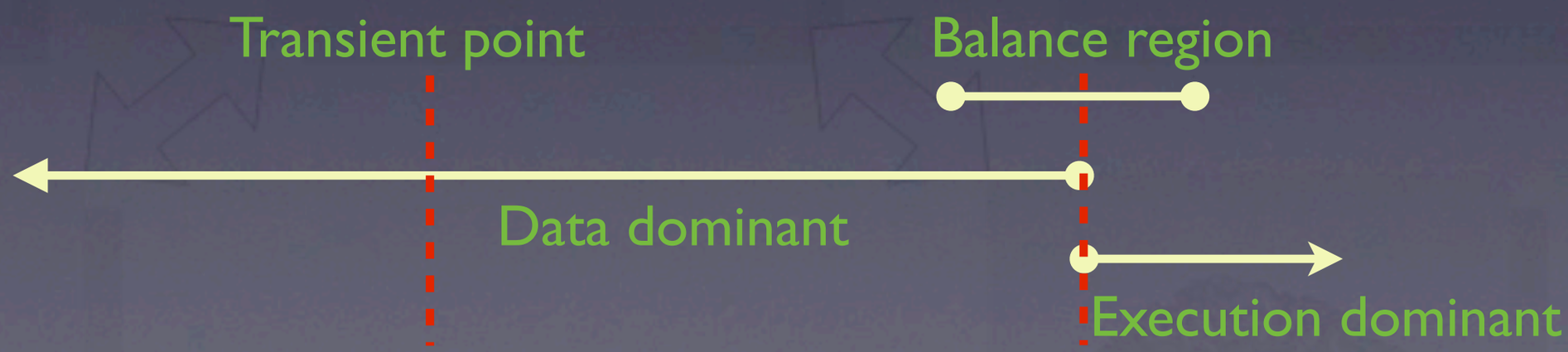


r = E/(U+D)	<<1	1	>>1
Function type	Transfer dominant	Balance	Processing dominant
Speed up benefit	Low	High (2)	Low

The system with one data transfer + one execution units

Ideal condition runtime





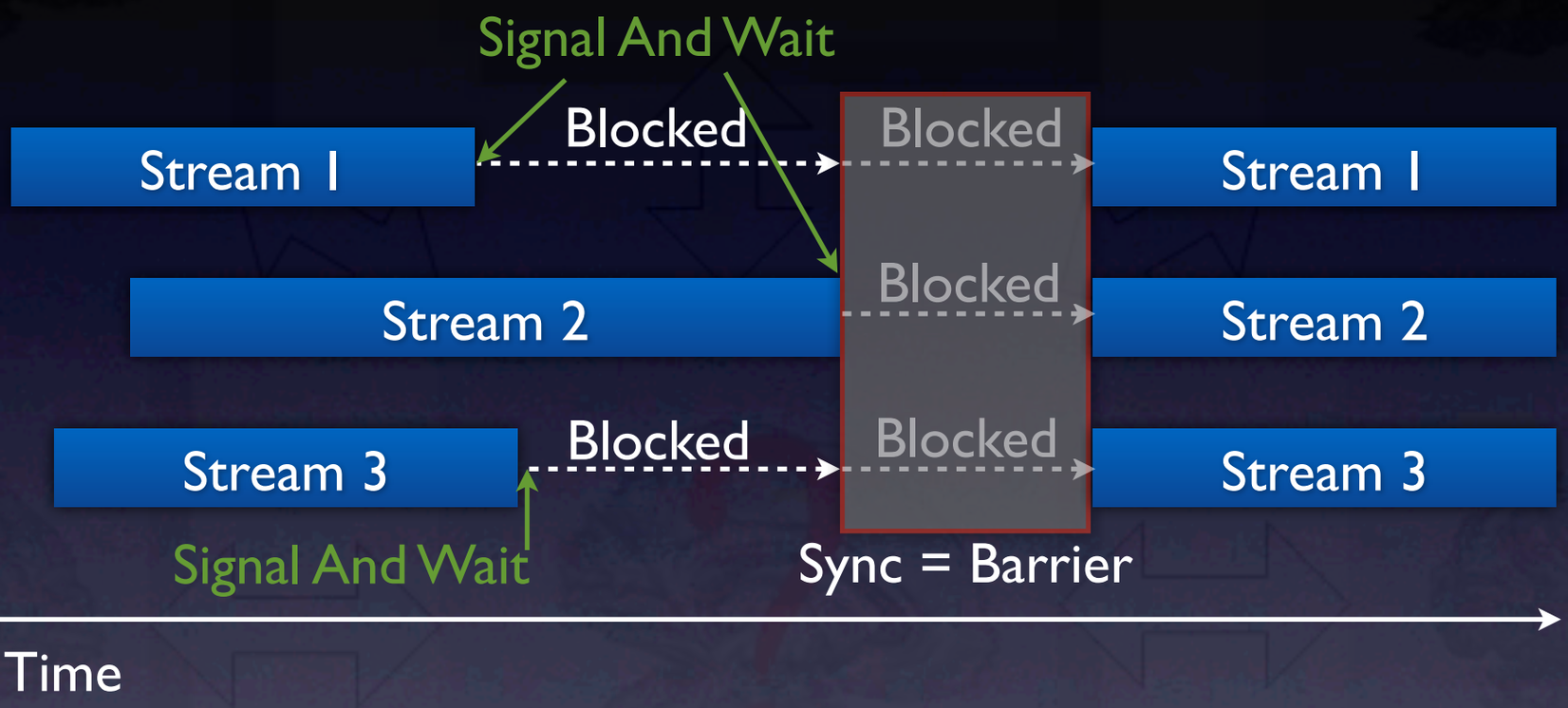
Outline

1. Motivation
2. Multi-image processing operators
3. Asynchronous streaming models
4. Degrading problem & solution
5. Conclusion

Asynchronous Processing in Practice: Degrading Conditions

Forced synchronization

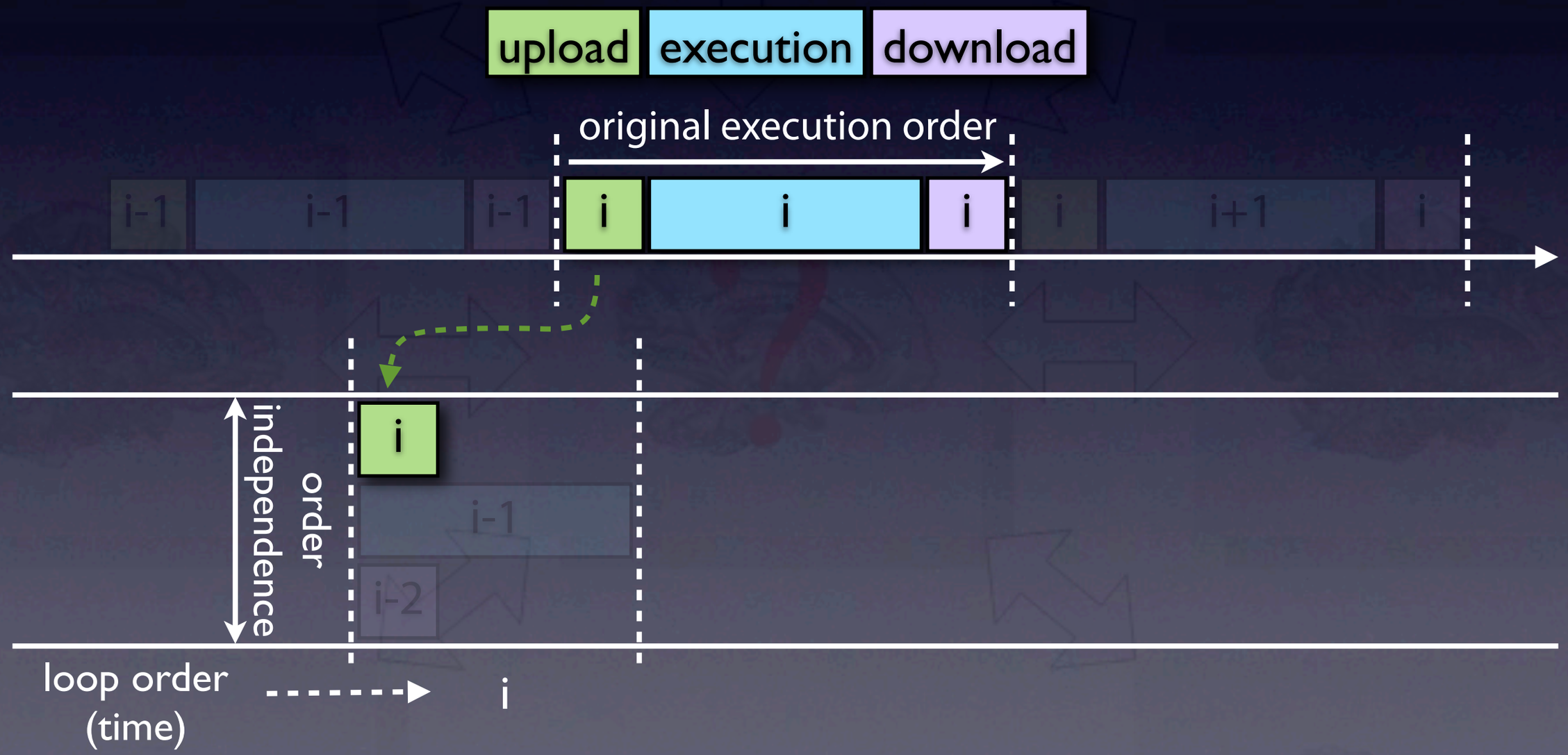
required to preserve semantic order of a program



- Reasons
  - Synchronization calls
  - Call to external functions without asynchronous support
  - Asynchronous mismatch
  - Cross-stream functions (in-stream synchronization)

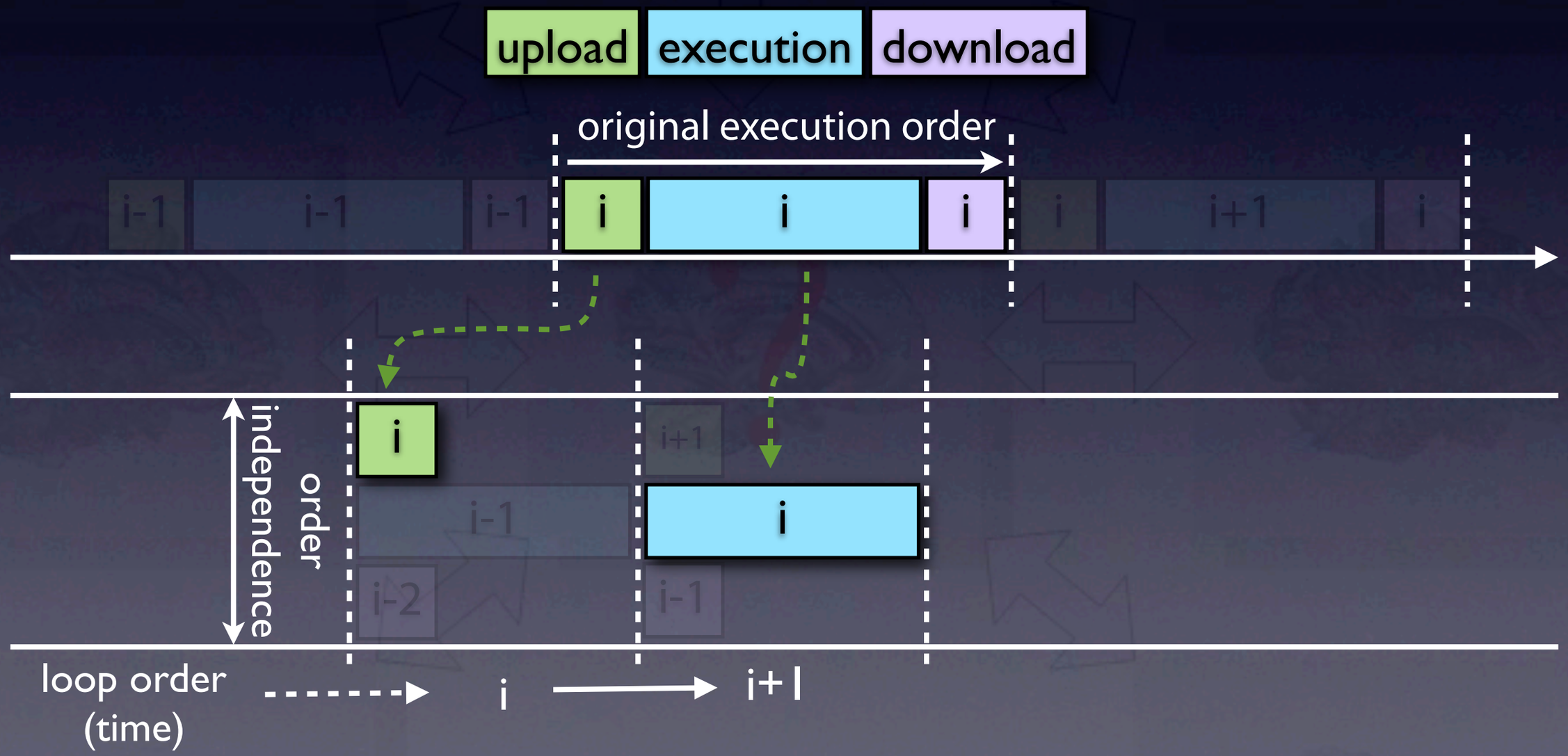
Order-independent Streaming Mode

- Reordering is available for stages of different images



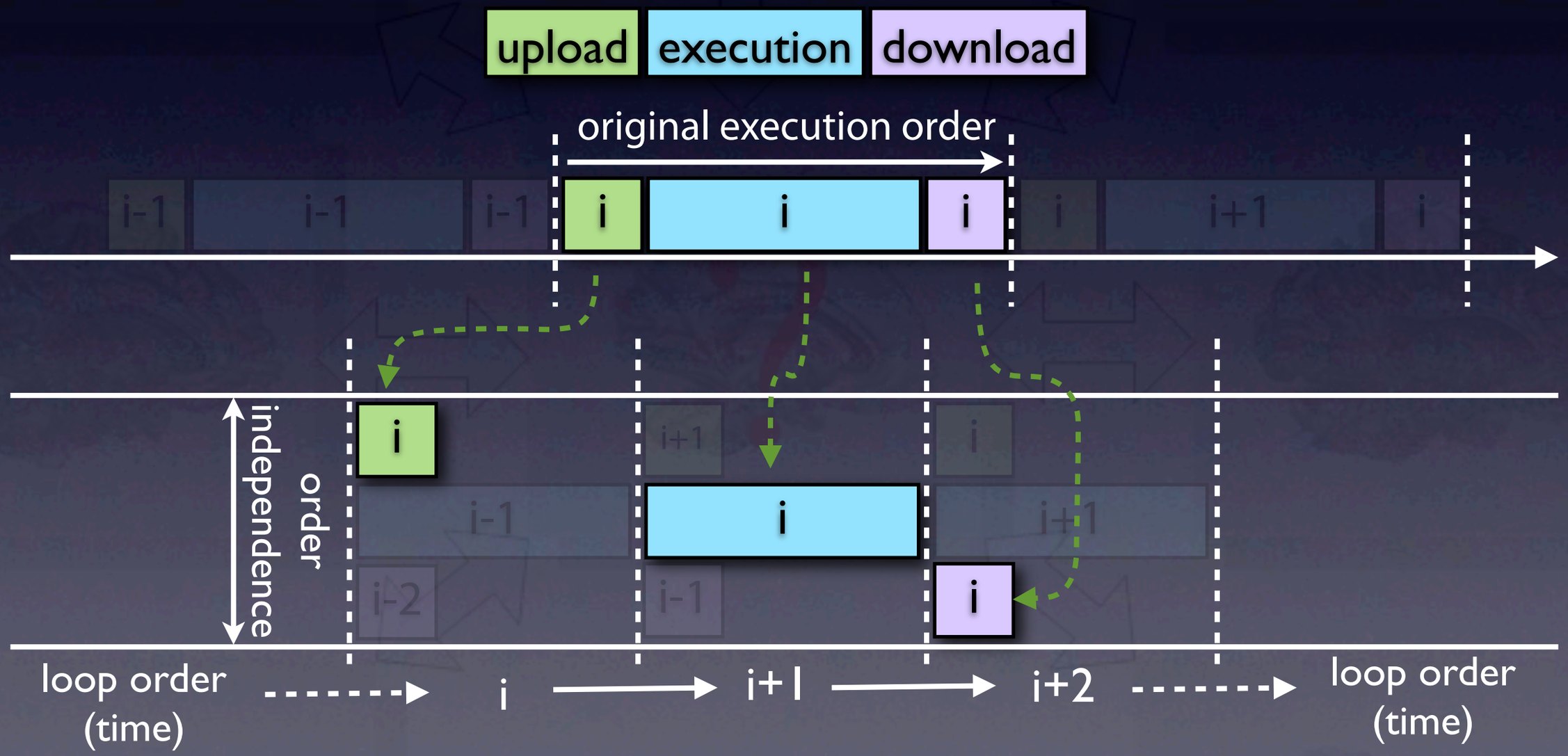
Order-independent Streaming Mode

- Reordering is available for stages of different images



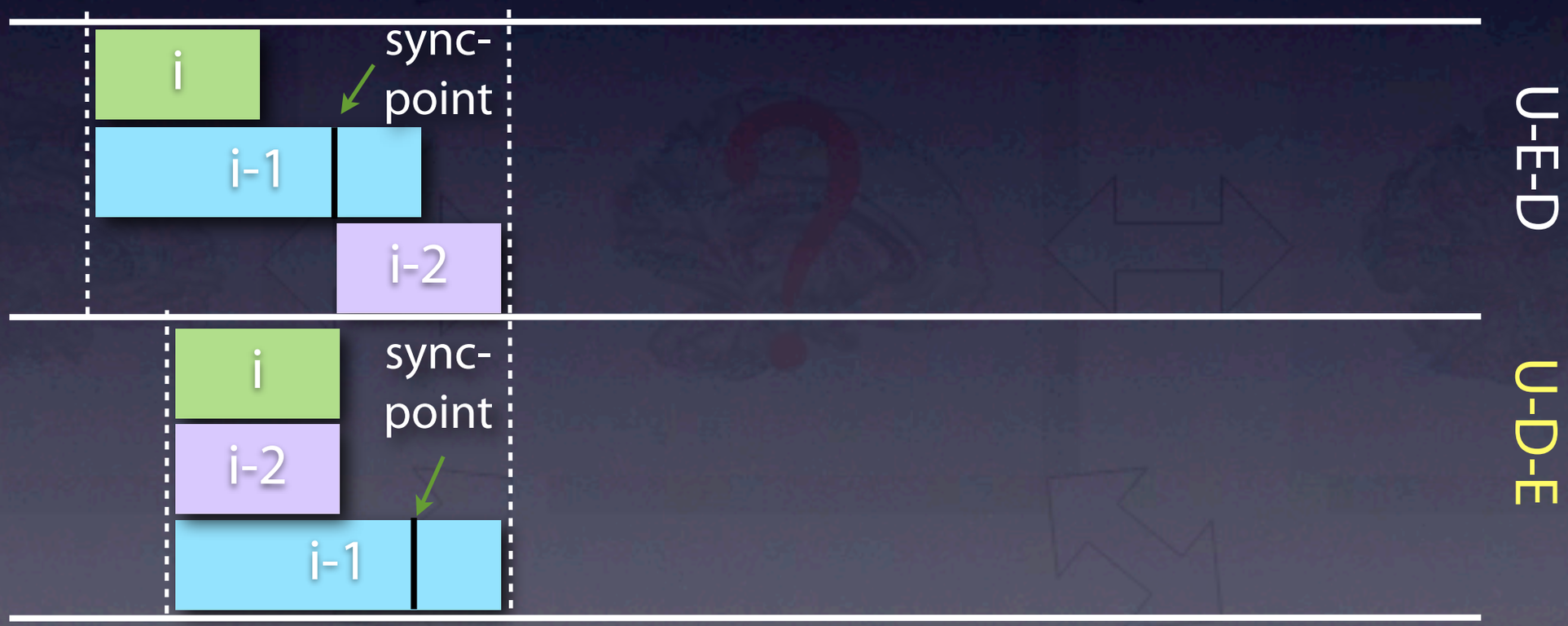
Order-independent Streaming Mode

- Reordering is available for stages of different images



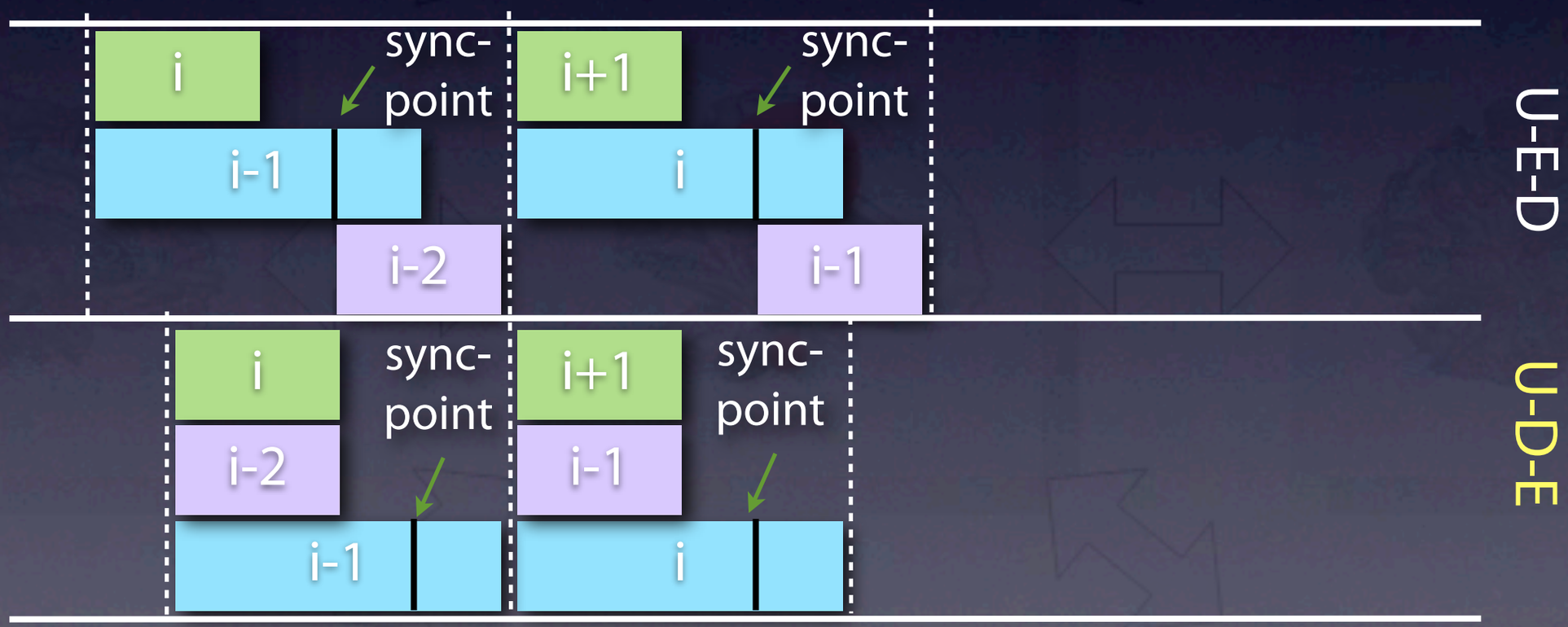
Solution for forced synchronization

- Reordering reduces penalty of forced synchronization in execution stage



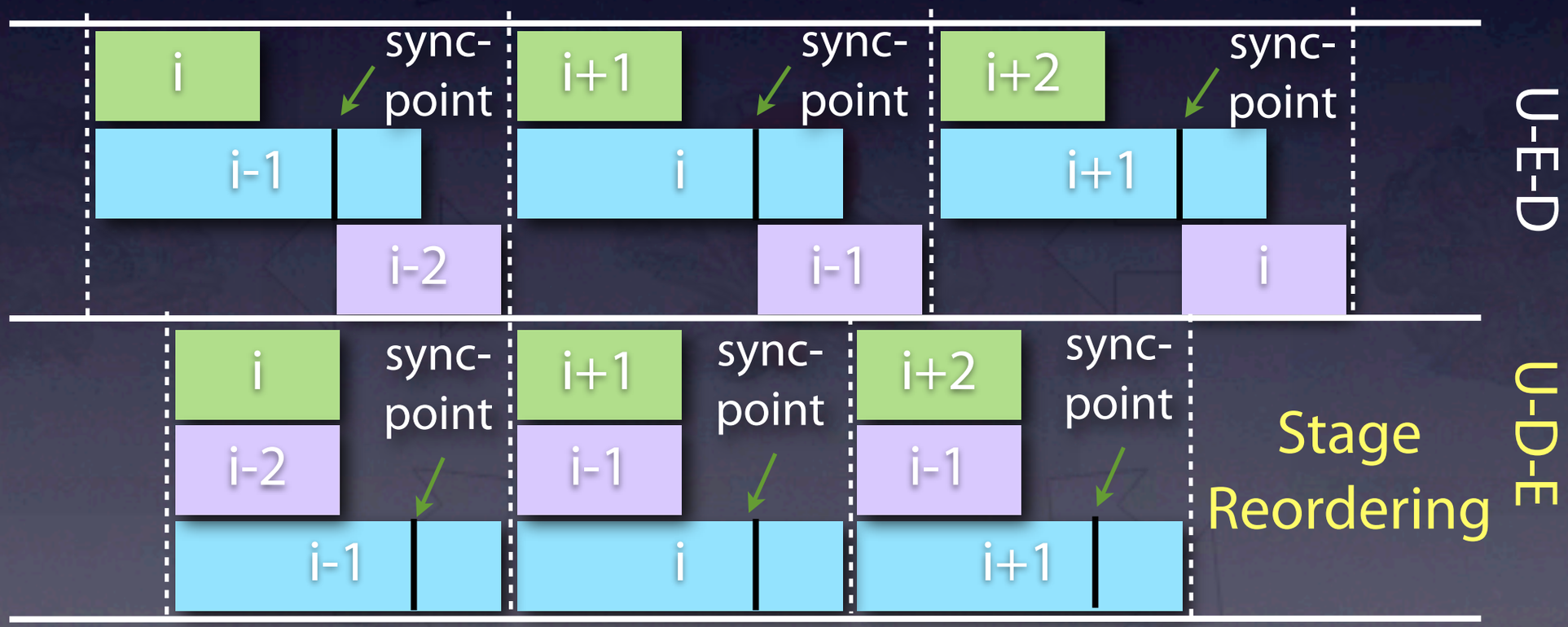
Solution for forced synchronization

- Reordering reduces penalty of forced synchronization in execution stage

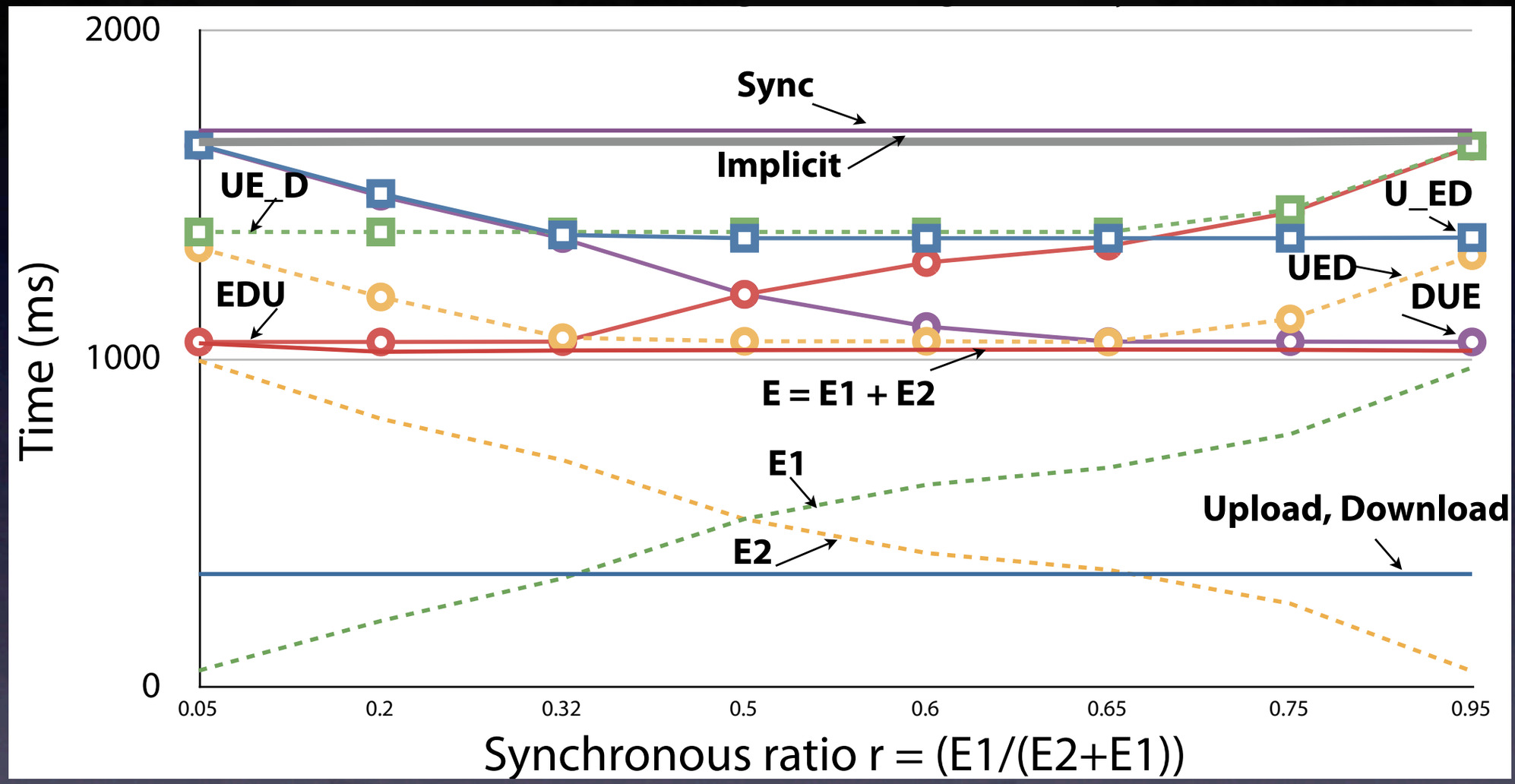


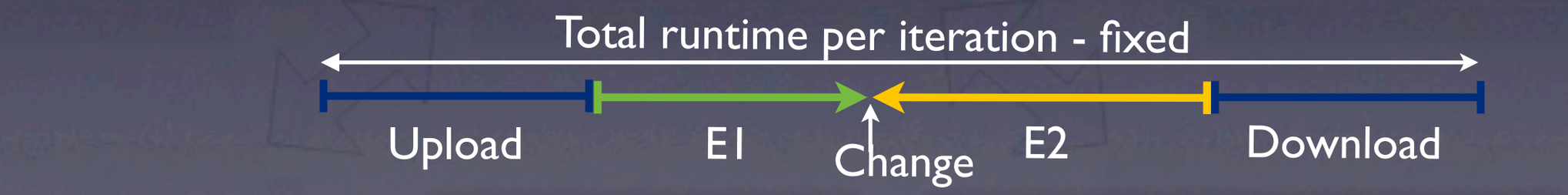
Solution for forced synchronization

- Reordering reduces penalty of forced synchronization in execution stage



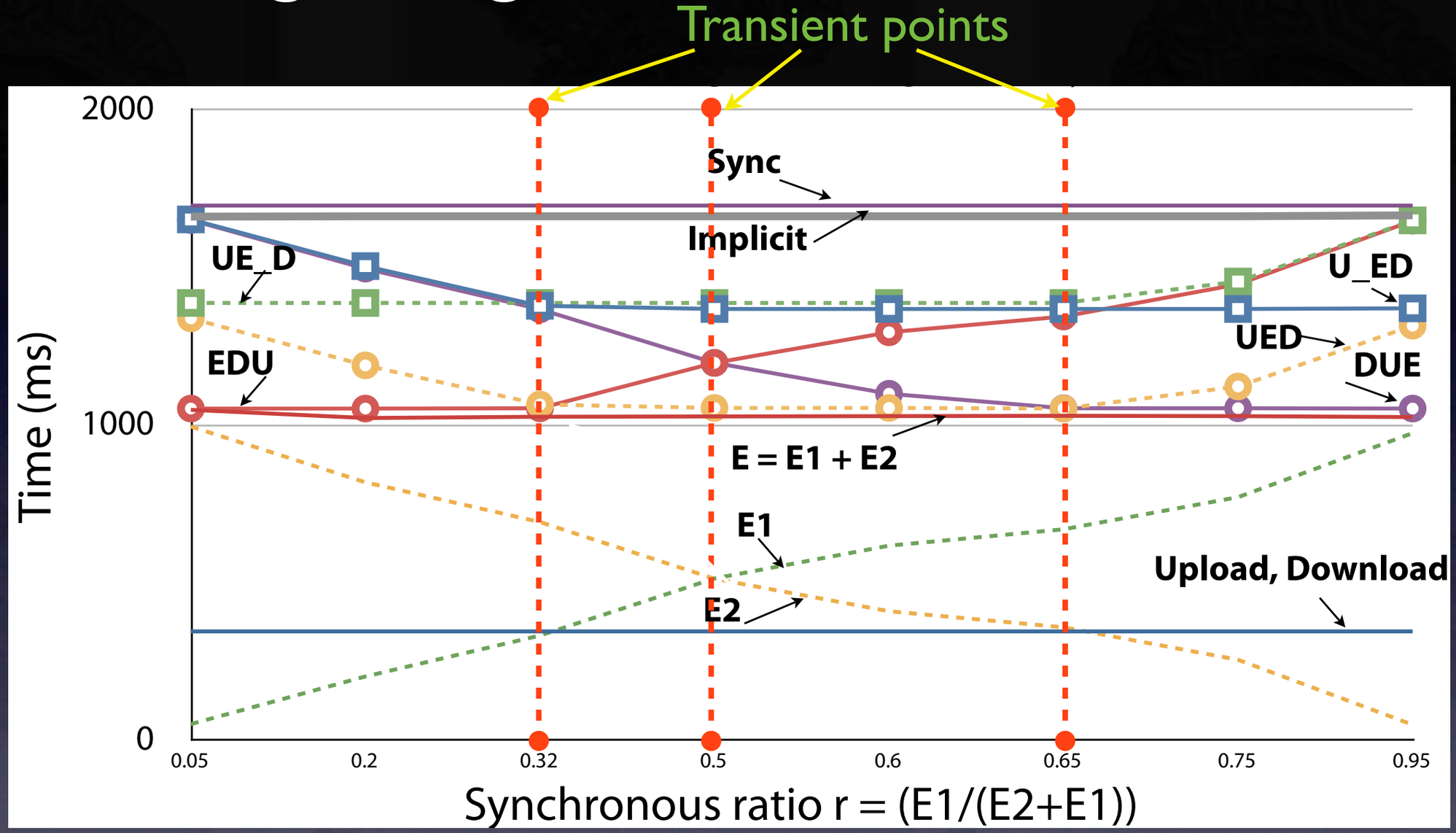
Degrading condition runtime





Synchronous
Implicit
U_ED
UE_D
Hardware-aware Exc
UED
DUE
EDU
Hardware-Independent Exc

Degrading condition runtime



E1 < TU	Tu<E1<E2	E1>E2>TD	E2>TD
EUD	UED	UED	DUE

Optimal strategy lookup table

Functions in practice

Function	U	E	D	Sync	Impl	Hrd-aware	Hrd-indp
Max	347	13	0	360	349	349	349
Energy	692	20	0	710	698	700	698
Averaging	347	20	11	378	360	363	361
Normalization	347	28	322	694	696	687	677
Gaussian	347	431	322	1099	735	770	678
Atlas	201446	213423	1359583	555204	NA	372567	340356

Functions in practice

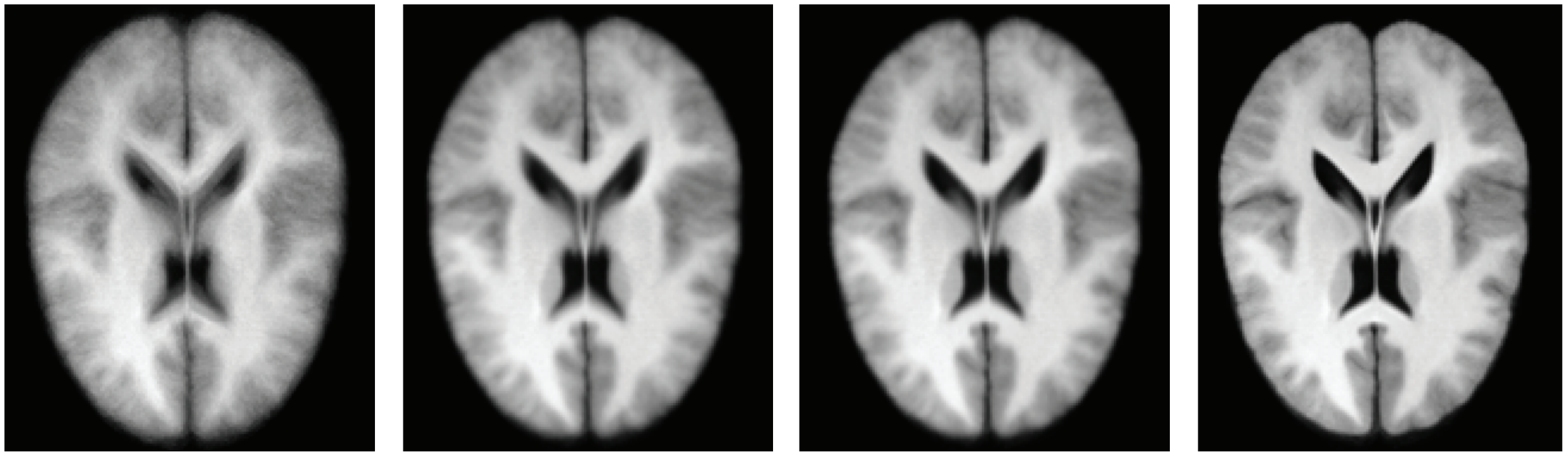
Function	U	E	D	Sync	Impl	Hrd-aware	Hrd-indp
Max	347	13	0	360	349	349	349
Energy	692	20	0	710	698	700	698
Averaging	347	20	11	378	360	363	361
Normalization	347	28	322	694	696	687	677
Gaussian	347	431	322	1099	735	770	678
Atlas	201446	213423	1359583	555204	NA	372567	340356

Functions in practice

Function	U	E	D	Sync	Impl	Hrd-aware	Hrd-indp
Max	347	13	0	360	349	349	349
Energy	692	20	0	710	698	700	698
Averaging	347	20	11	378	360	363	361
Normalization	347	28	322	694	696	687	677
Gaussian	347	431	322	1099	735	770	678
Atlas	201446	213423	1359583	555204	NA	372567	340356

Note: Full Out-Of-Core Atlas Construction takes 1h40 minutes with Async. execution and 2 hours with Sync. execution to finish

Approximate Atlas versus Complete Atlas

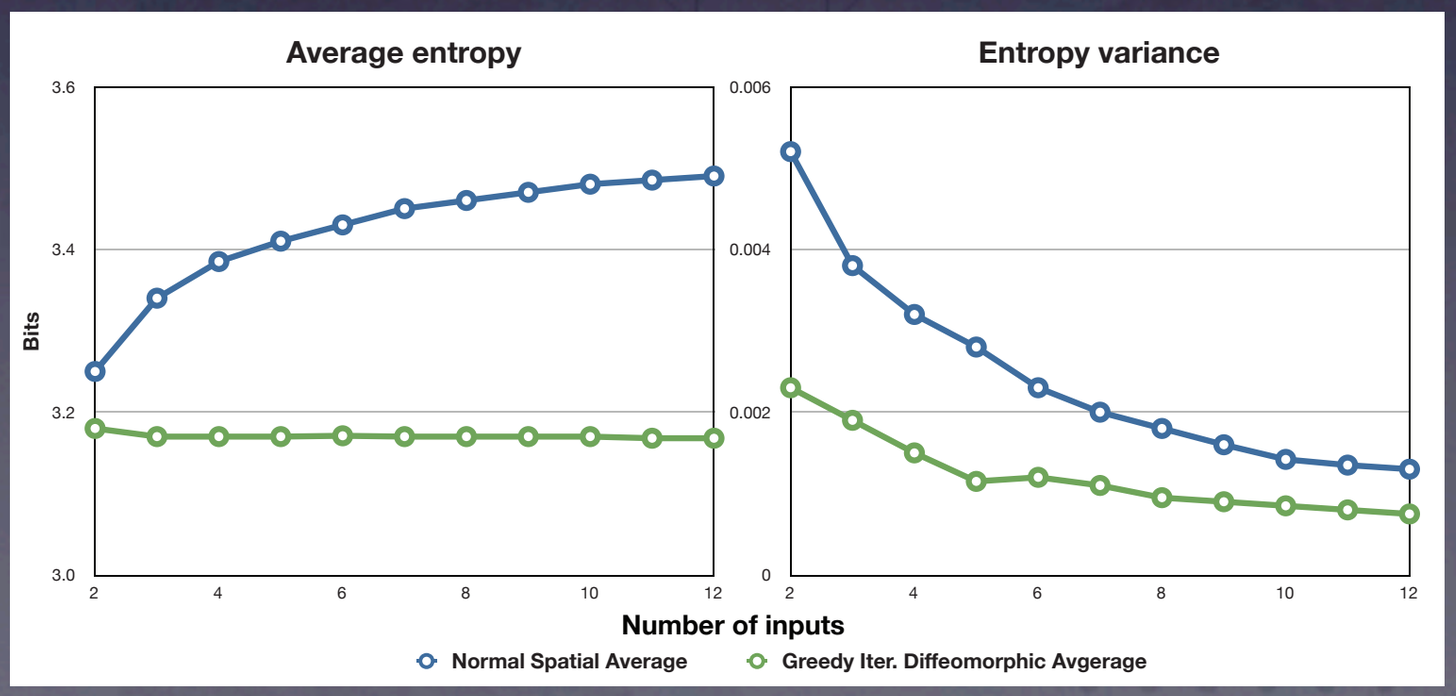


a. Intensity average image

b. Random-permuation regression atlas construction with cohort = 3

c. Random-permuation regression atlas construction with cohort = 9

d. Full-diffeomorphic atlas construction Number of images 156



Permutation regression

- Number of tests = 100
- Cohort = 3 takes 40 minutes
- Cohort = 9 takes 2 hours

Exact out-of-core computation takes 6 minutes

Conclusion

- This paper presents
  - Multi-image processing framework
    - Multi-image operators
  - Efficient asynchronous execution model
    - Hardware independent model is optimal
  - Degrading problem and solution based on reordering strategy

- Future work
  - Reduce the constraints
  - Out-of-core processing for general data structure

Software package will be available to download at
http://www.sci.utah.edu/software/13/370-atlaswerks.html

Acknowledgment

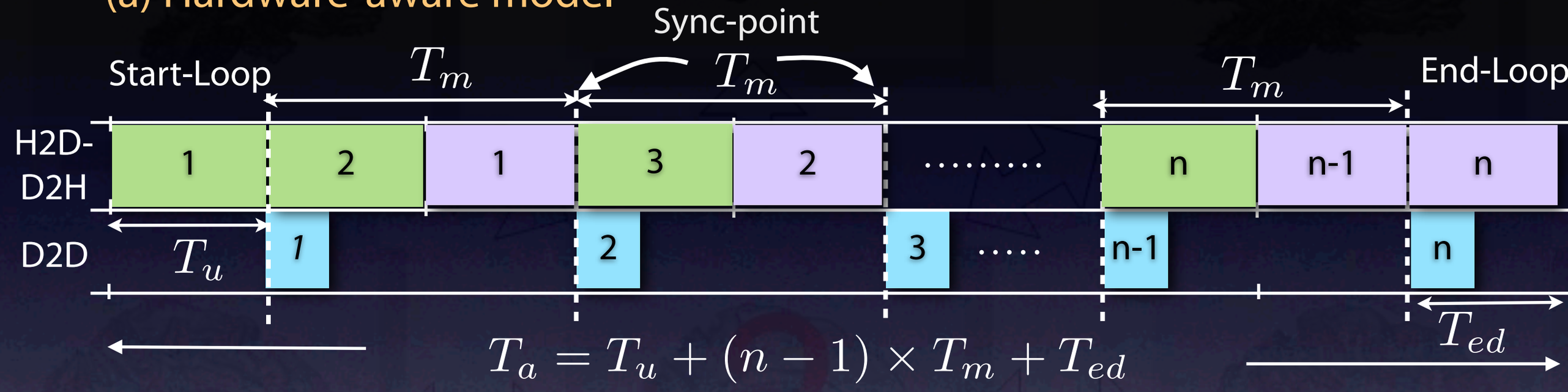
- Grants:
  - 5R01EB007688, MH064065, P41 RR023953 and NSF grant CNS-07511152
  - Vietnam Education Foundation

- Thank you for attending my talk

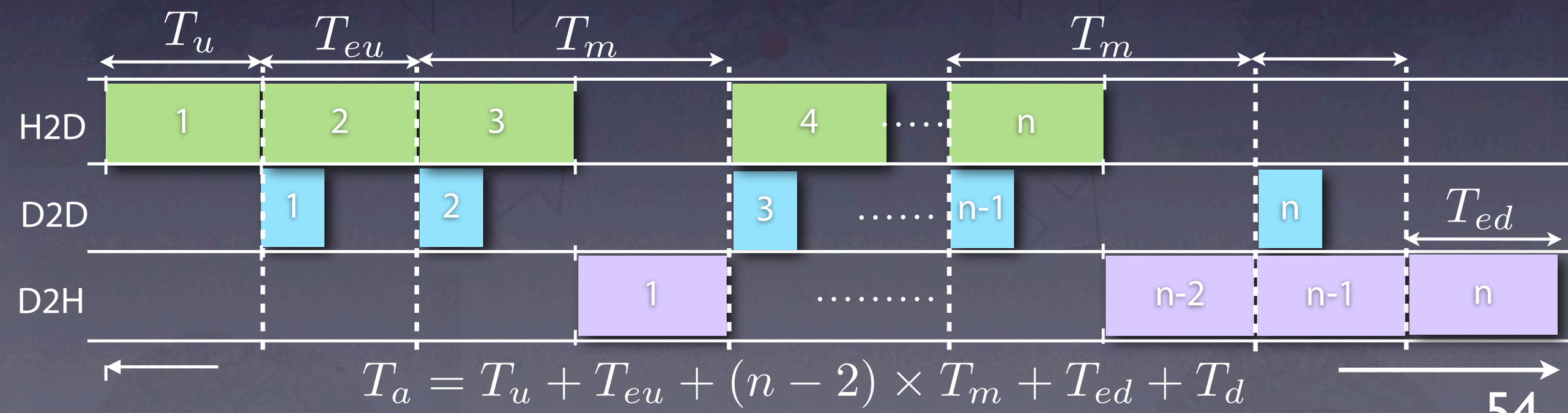
Question and Answer

Hardware independent optimality

(a) Hardware-aware model



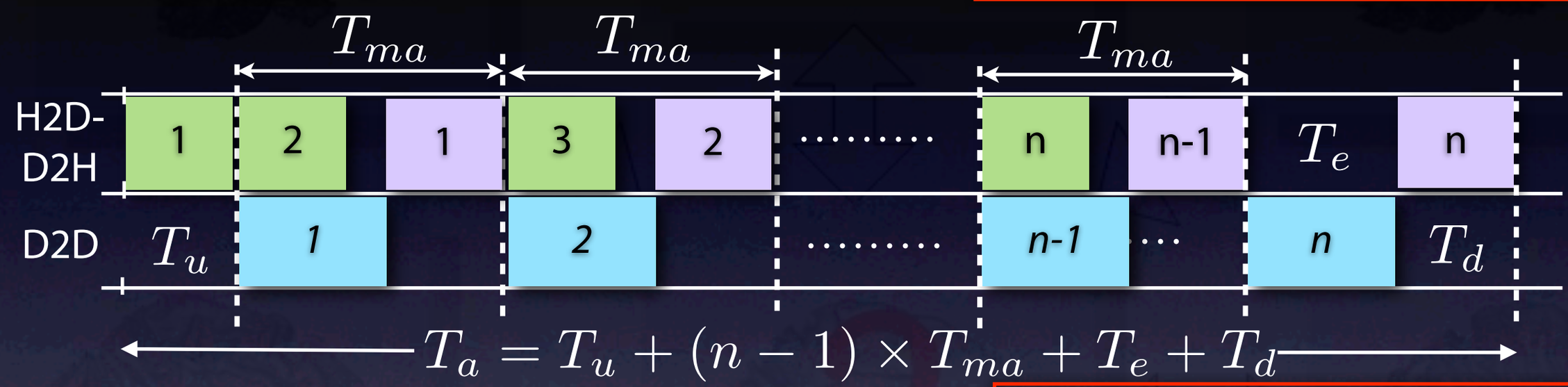
(b) Hardware-independent model



When hardware independent is faster than hardware aware

(a) Hardware-aware model

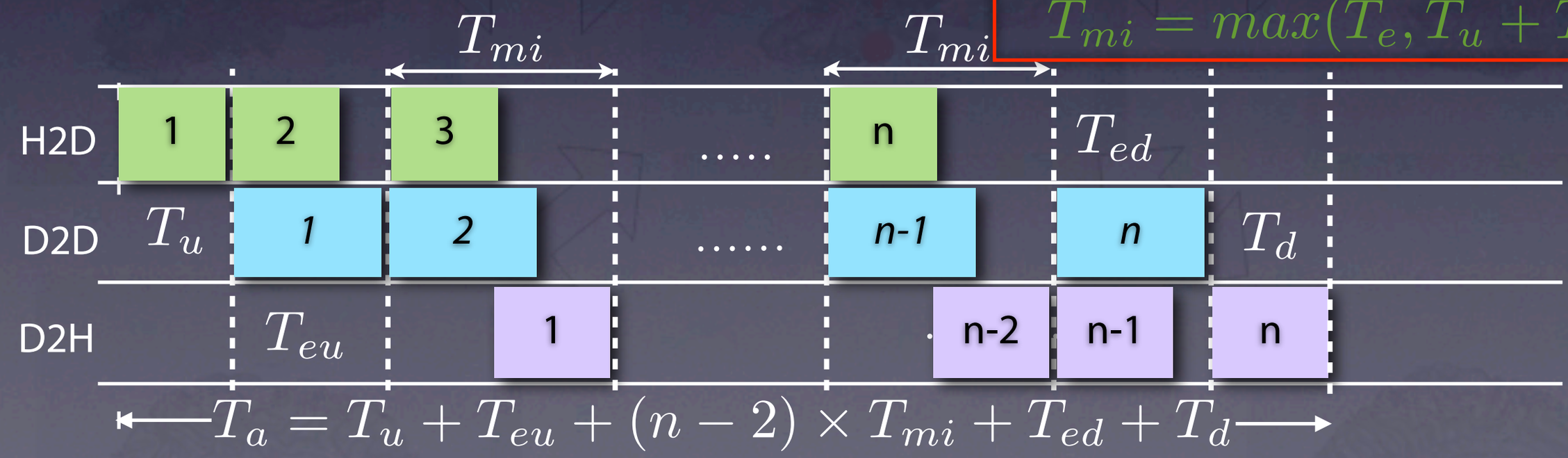
Consider $T_e > max(T_u, T_d)$



(b) Hardware-independent model

$T_{ma} = max(T_u, T_e + T_d) >$

$T_{mi} = max(T_e, T_u + T_d)$

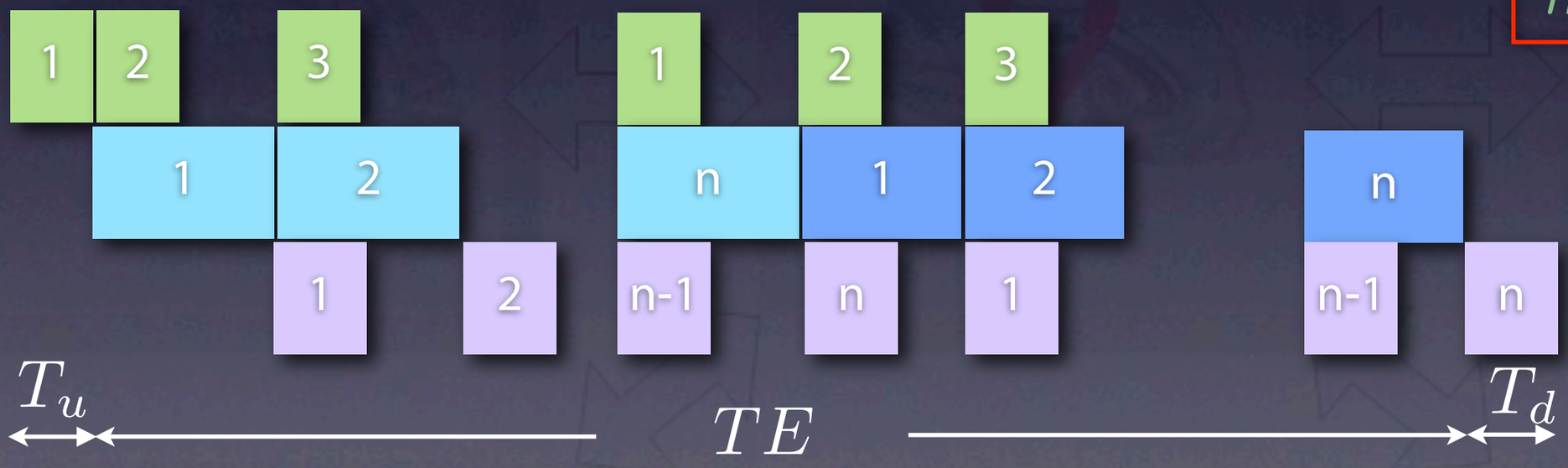


When out-of-core is faster than in-core

a. In-core execution (without asynchronous execution)

b. Out-of-core using asynchronous execution

$$max(T_u, T_d) < T_{e1}$$
$$max(T_u, T_d) < T_{e2}$$



- Reasons
  - The data transfer can be hidden completely by execution

Atlas construction algorithm

1: **Input** : N volume inputs
2: **Output**: Template atlas volume
3: **for** $k = 1$ to max_iters **do**
4: Fix images I_i^k, compute the template $\hat{I^k} = \frac{1}{N} \frac{\sum_{i=1}^{N} I_i^k w_i}{\sum_{i=1}^{N} w_i}$
5: **for** $i = 1$ to N **do** {loop over the images}
6: Fix the template $\hat{I^k}$, solve pairwise-matching problem between I_i^k and $\hat{I^k}$
7: Update deformed image I_i^k with current velocity
8: **end for**
9: **end for**

Algorithm 1: Atlas construction framework

1: **Input** : N input images
2: **Output**: N processed output images
3: **for** $k = 1$ to N **do**
4: Upload the k-th image from the storage device to the processing device
5: Process the input in-core on the processing device
6: Download the output image back to the storage device
7: **end for**

Algorithm 2: Synchronous out-of-core MIMO operators

1: **Input** : N input volumes, device input buffers $d_i[3]$ and device input buffers $d_o[3]$
2: **Output**: N processed output volumes
3: **for** $k = 1$ to $N+2$ **do**
4: **if** $k <= N$ **then**
5: Load the data $iImg[k]$ from storage device to device buffer $d_i[k\%3]$ on the*H2D* stream
6: **end if**
7: **if** $k > 1$ and $k-1 <= N$ **then**
8: Apply the operator on device buffer $d_o[(k-1)\%3] = oper(d_i[(k-1)\%3])$ on *D2D* stream
9: **end if**
10: **if** $k > 2$ and $k-2 <= N$ **then**
11: Write output $d_o[(k-2)\%3]$ to the storage device $oImg[(k-2)]$ on the *D2H* stream
12: **end if**
13: **Synchronize streams**
14: **end for**

Algorithm 5: Explicit pipelining MIMO operator