HAJIM
SCHOOL OF ENGINEERING
& APPLIED SCIENCES
COMPUTER SCIENCE

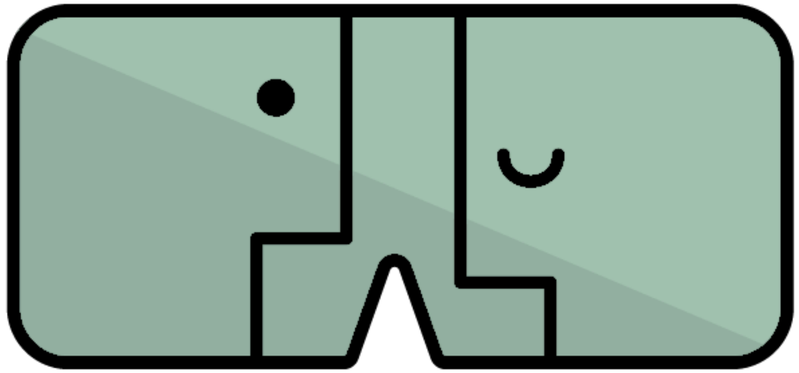
The Next Quintillion Pixels:

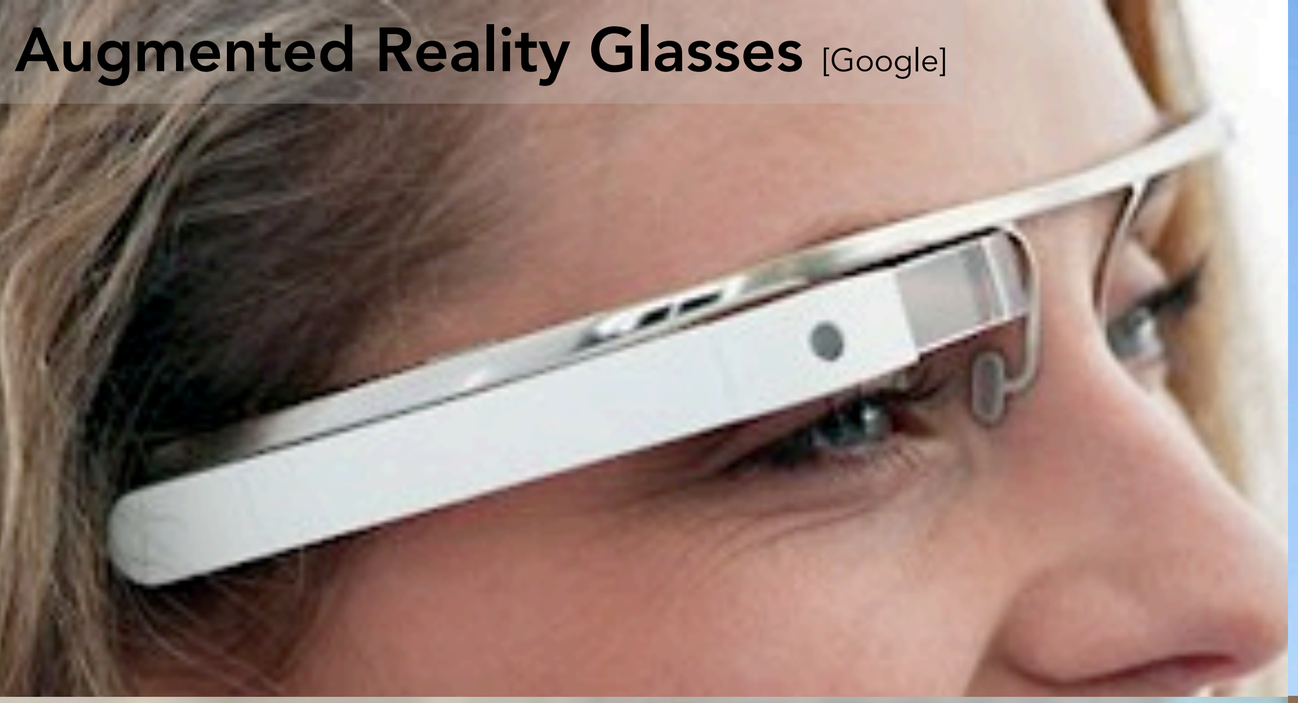
Architecting Next-Generation Mobile Visual Computing Systems

Yuhao Zhu

Assistant Professor — Horizon Lab
Department of Computer Science
University of Rochester

http://horizon-lab.org

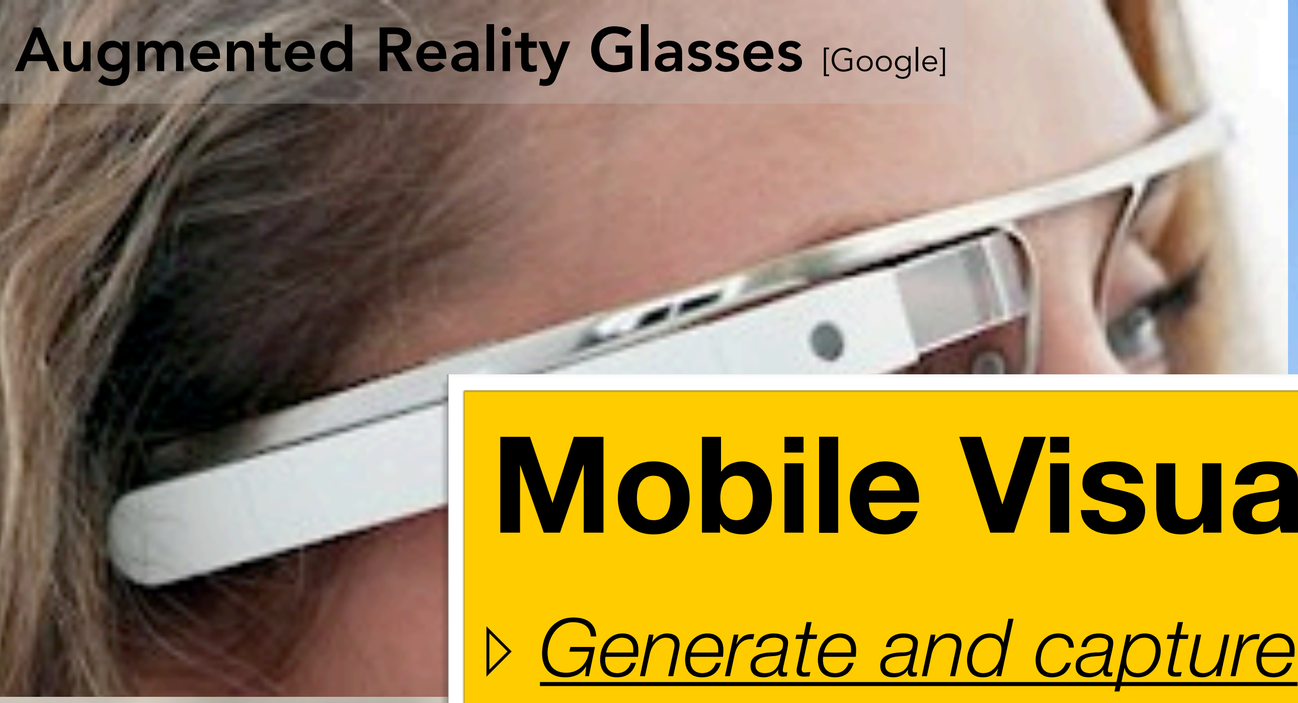



Augmented Reality Glasses [Google]


Autonomous (Aerial) Machines [DJI]

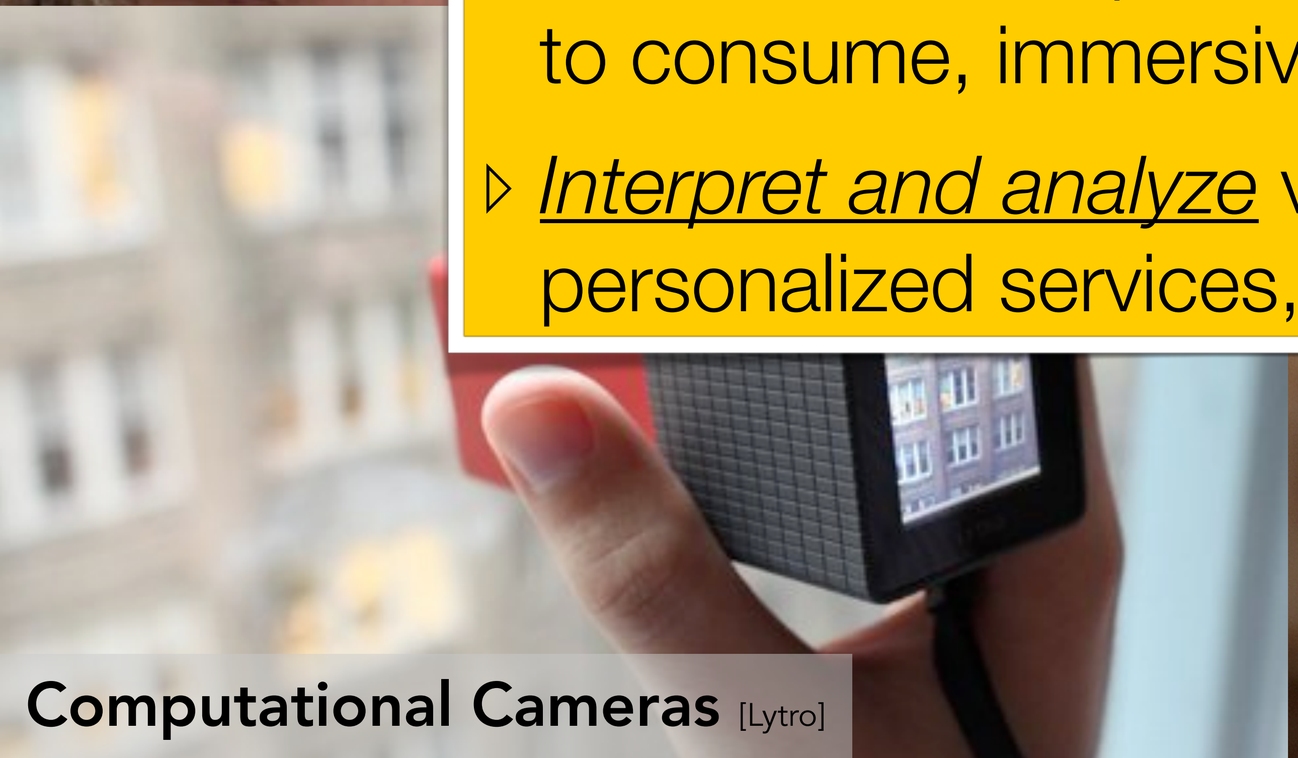

Computational Cameras [Lytro]


360° VR Camera Rigs [Facebook]



Mobile Visual Computing

- *Generate and capture* visual data for human to consume, immersively;
- *Interpret and analyze* visual data to provide personalized services, intelligently.

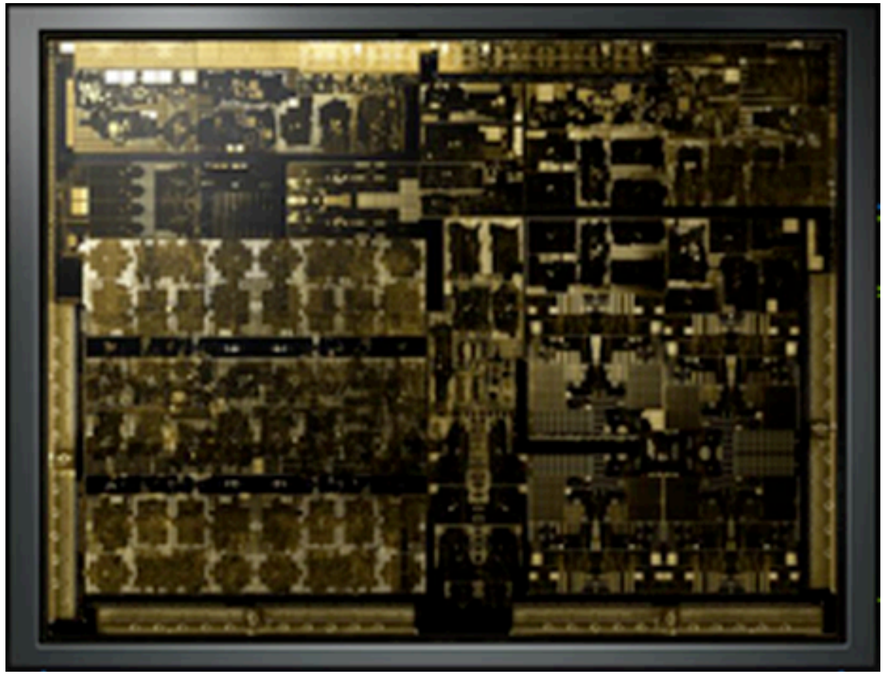


What does it take to architect next-generation visual computing systems that are orders of magnitude faster and more energy-efficient than what's currently out there?

Exploiting Synergies Across Visual Computing Domains

https://www.computerbase.de/2018-01/nvidia-drive-xavier-soc-sample/

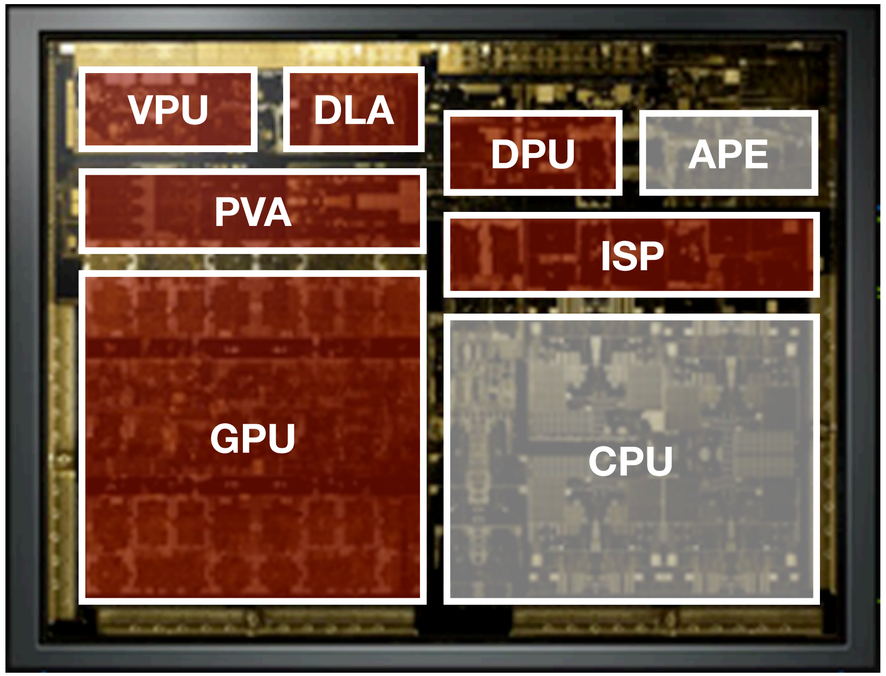
Exploiting Synergies Across Visual Computing Domains



Nvidia Xavier SoC (2018)
Illustration only.

https://www.computerbase.de/2018-01/nvidia-drive-xavier-soc-sample/

Exploiting Synergies Across Visual Computing Domains



Nvidia Xavier SoC (2018)
Illustration only.

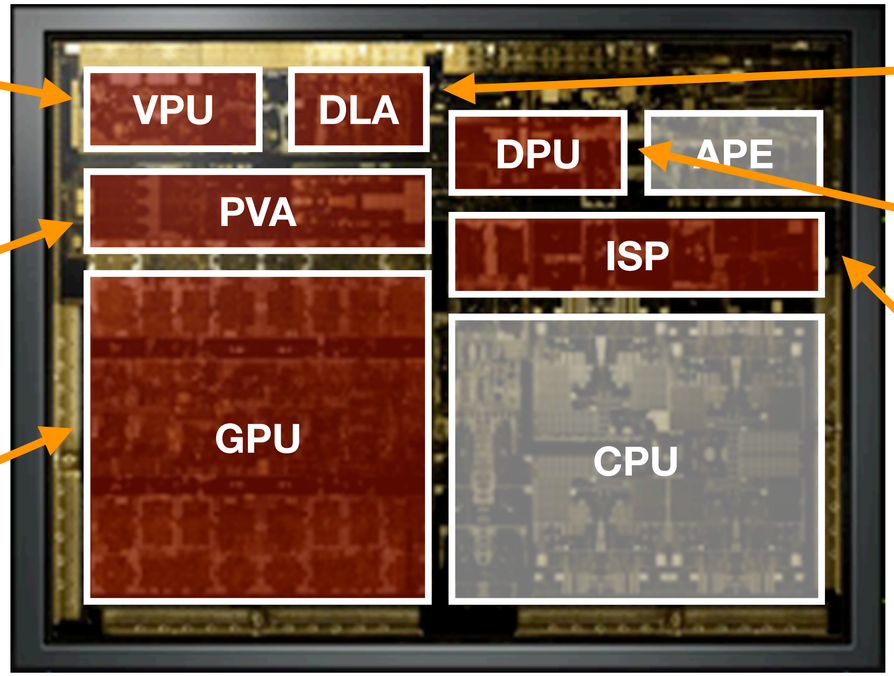
https://www.computerbase.de/2018-01/nvidia-drive-xavier-soc-sample/

Exploiting Synergies Across Visual Computing Domains

Video Processing Unit
(Video encode/decode)

Vision Accelerator
(Computer vision)

Graphics Processing Unit
(Graphics, data-parallel algo.)



Deep Learning Accelerator
(Deep neural networks)

Display Processing Unit
(Scale, rotate, composition, etc.)

Image Signal Processor
(Convert camera sensor data
to frame pixels)

Nvidia Xavier SoC (2018)
Illustration only.

https://www.computerbase.de/2018-01/nvidia-drive-xavier-soc-sample/

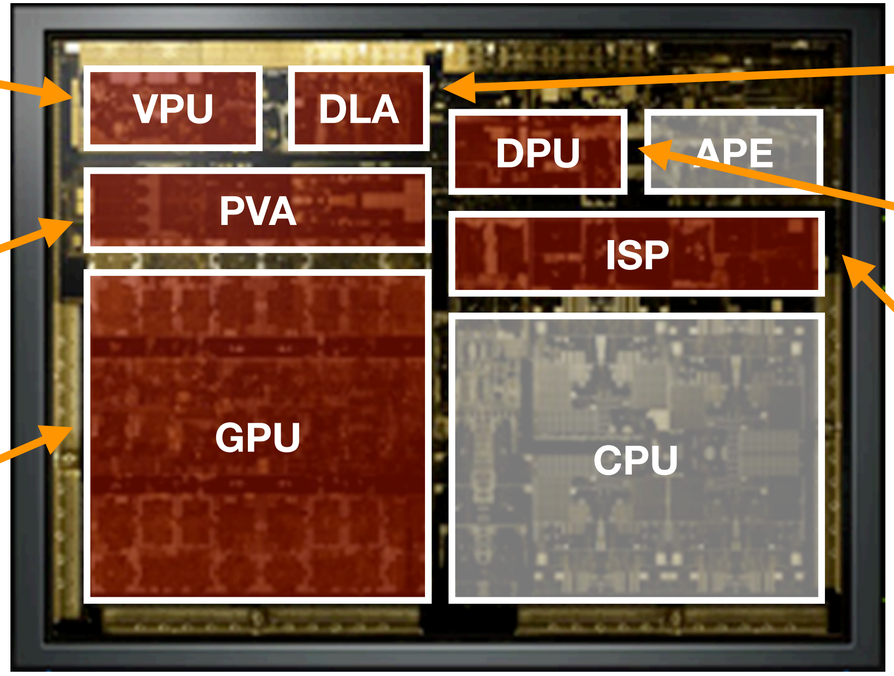
Exploiting Synergies Across Visual Computing Domains

Today: Stitching sub-systems designed for individually domains

Video Processing Unit
(Video encode/decode)

Vision Accelerator
(Computer vision)

Graphics Processing Unit
(Graphics, data-parallel algo.)



Deep Learning Accelerator
(Deep neural networks)

Display Processing Unit
(Scale, rotate, composition, etc.)

Image Signal Processor
(Convert camera sensor data
to frame pixels)

Nvidia Xavier SoC (2018)
Illustration only.

https://www.computerbase.de/2018-01/nvidia-drive-xavier-soc-sample/

Exploiting Synergies Across Visual Computing Domains

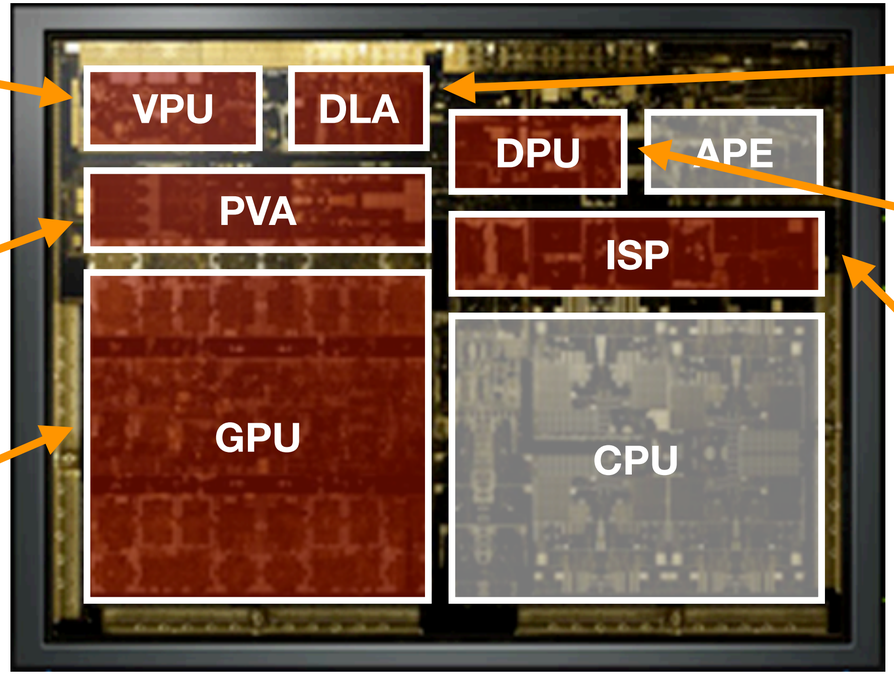
Today: Stitching sub-systems designed for individually domains

- But visual applications inherently exercise tasks from different domains

Video Processing Unit
(Video encode/decode)

Vision Accelerator
(Computer vision)

Graphics Processing Unit
(Graphics, data-parallel algo.)



Deep Learning Accelerator
(Deep neural networks)

Display Processing Unit
(Scale, rotate, composition, etc.)

Image Signal Processor
(Convert camera sensor data to frame pixels)

Nvidia Xavier SoC (2018)
Illustration only.

https://www.computerbase.de/2018-01/nvidia-drive-xavier-soc-sample/

Exploiting Synergies Across Visual Computing Domains

Today: Stitching sub-systems designed for individually domains

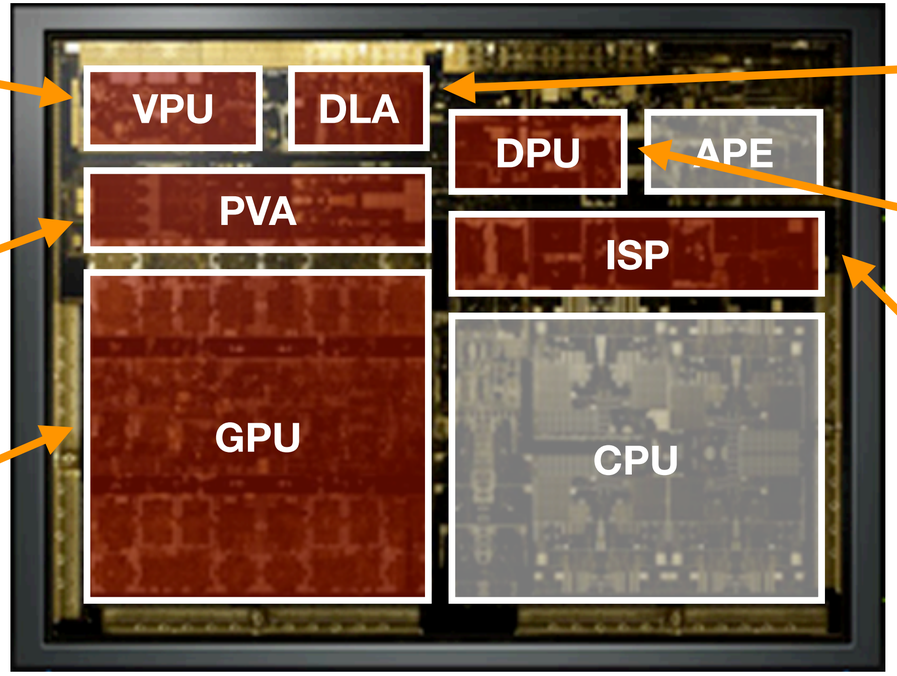
- But visual applications inherently exercise tasks from different domains

Key: Exploiting synergies across different domains

Video Processing Unit
(Video encode/decode)

Vision Accelerator
(Computer vision)

Graphics Processing Unit
(Graphics, data-parallel algo.)



Deep Learning Accelerator
(Deep neural networks)

Display Processing Unit
(Scale, rotate, composition, etc.)

Image Signal Processor
(Convert camera sensor data to frame pixels)

Nvidia Xavier SoC (2018)
Illustration only.

https://www.computerbase.de/2018-01/nvidia-drive-xavier-soc-sample/

Exploiting Synergies Across Visual Computing Domains

Today: Stitching sub-systems designed for individually domains

- But visual applications inherently exercise tasks from different domains

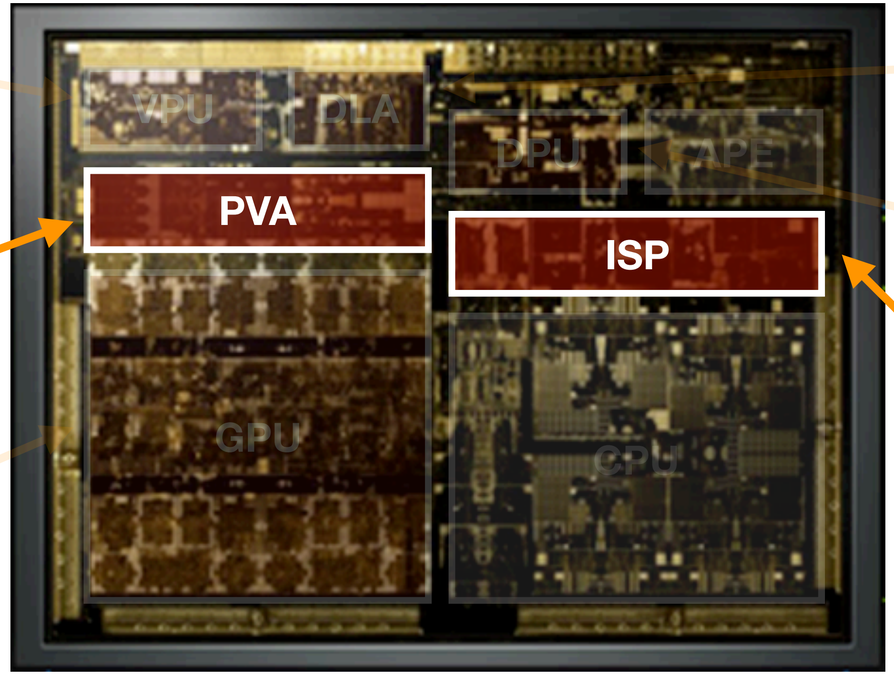
Key: Exploiting synergies across different domains

- This talk: co-designing imaging with vision [ISCA 2018, MICRO 2019]

Video Processing Unit
(Video encode/decode)

Vision Accelerator
(Computer vision)

Graphics Processing Unit
(Graphics, data-parallel algo.)



Deep Learning Accelerator
(Deep neural networks)

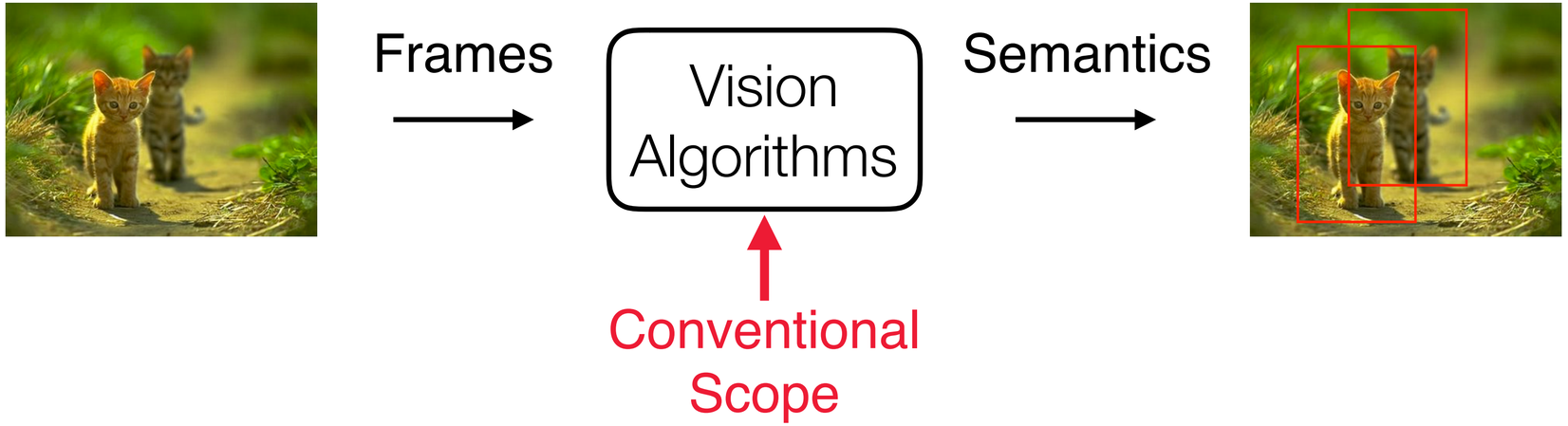
Display Processing Unit
(Scale, rotate, composition, etc.)

Image Signal Processor
(Convert camera sensor data to frame pixels)

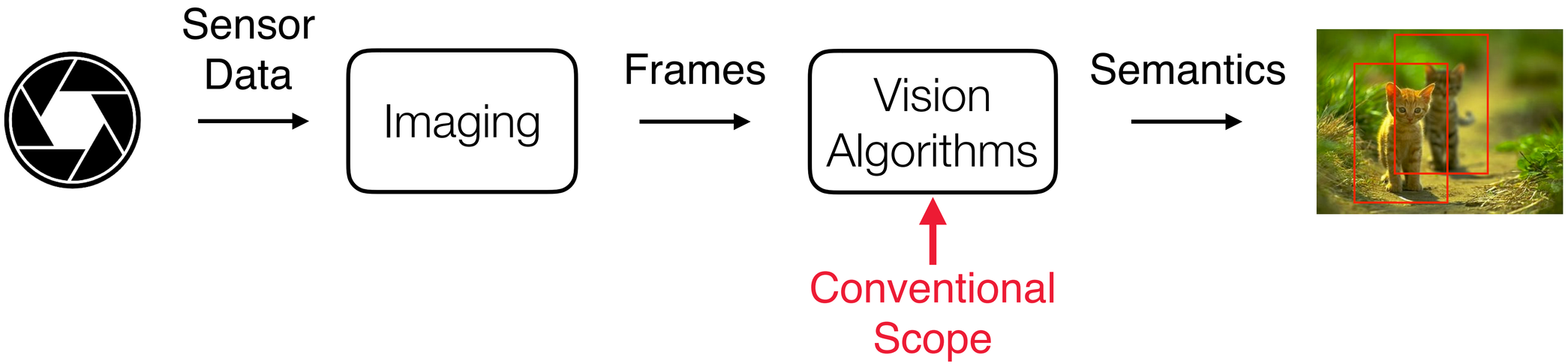
Nvidia Xavier SoC (2018)
Illustration only.

https://www.computerbase.de/2018-01/nvidia-drive-xavier-soc-sample/

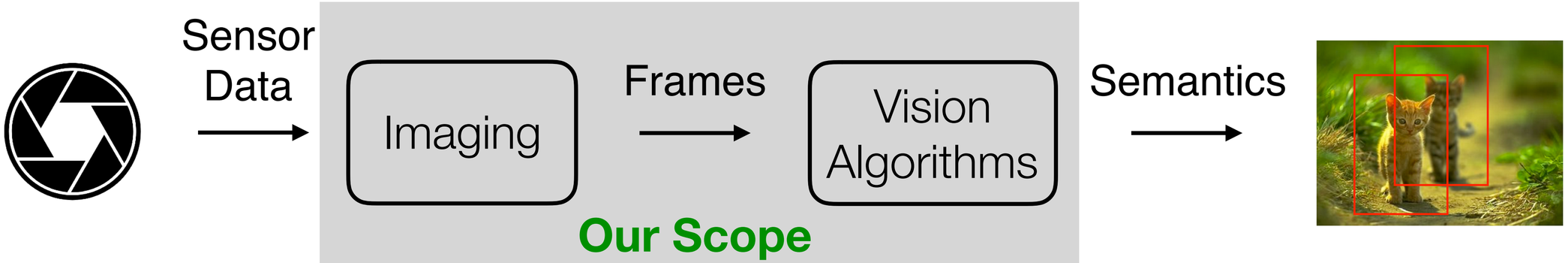
Co-Designing Imaging with Vision



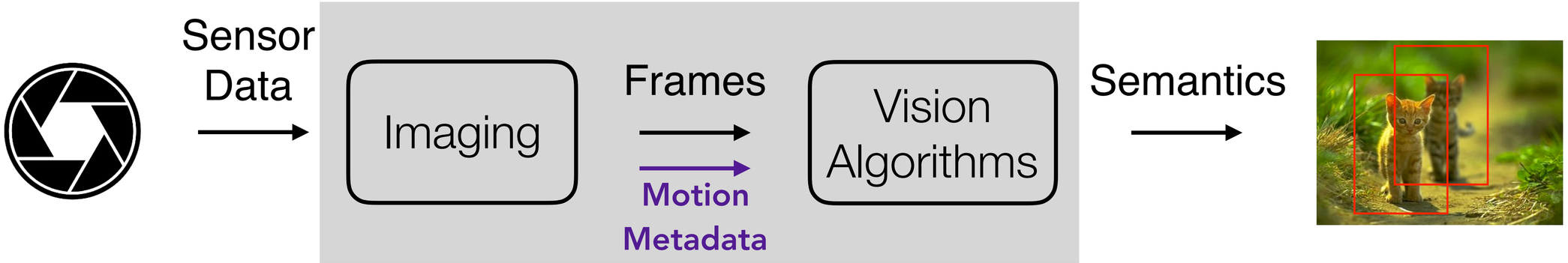
Co-Designing Imaging with Vision



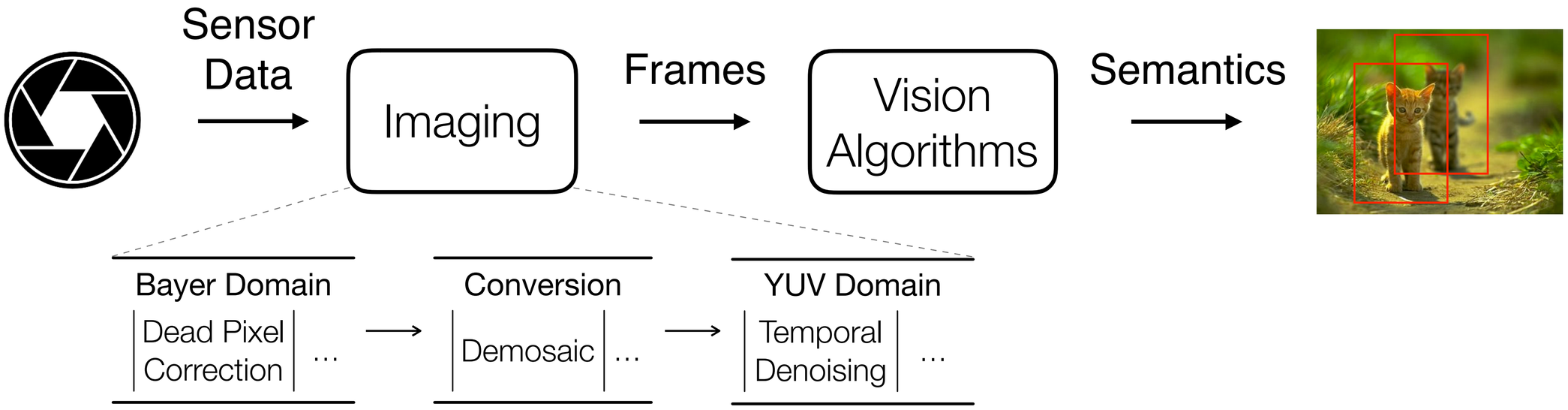
Co-Designing Imaging with Vision



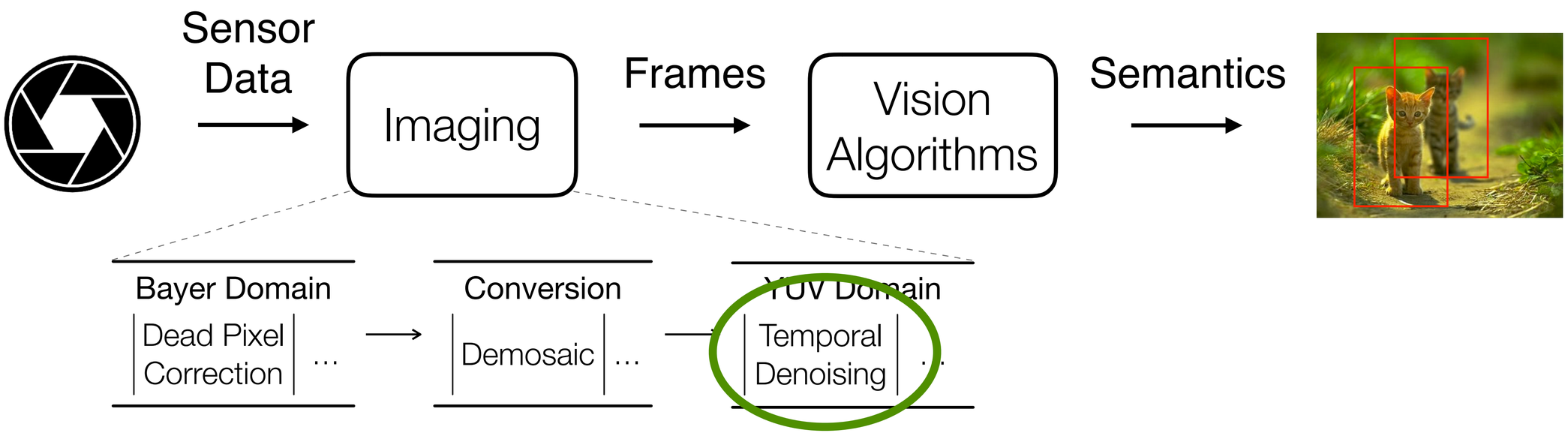
Co-Designing Imaging with Vision



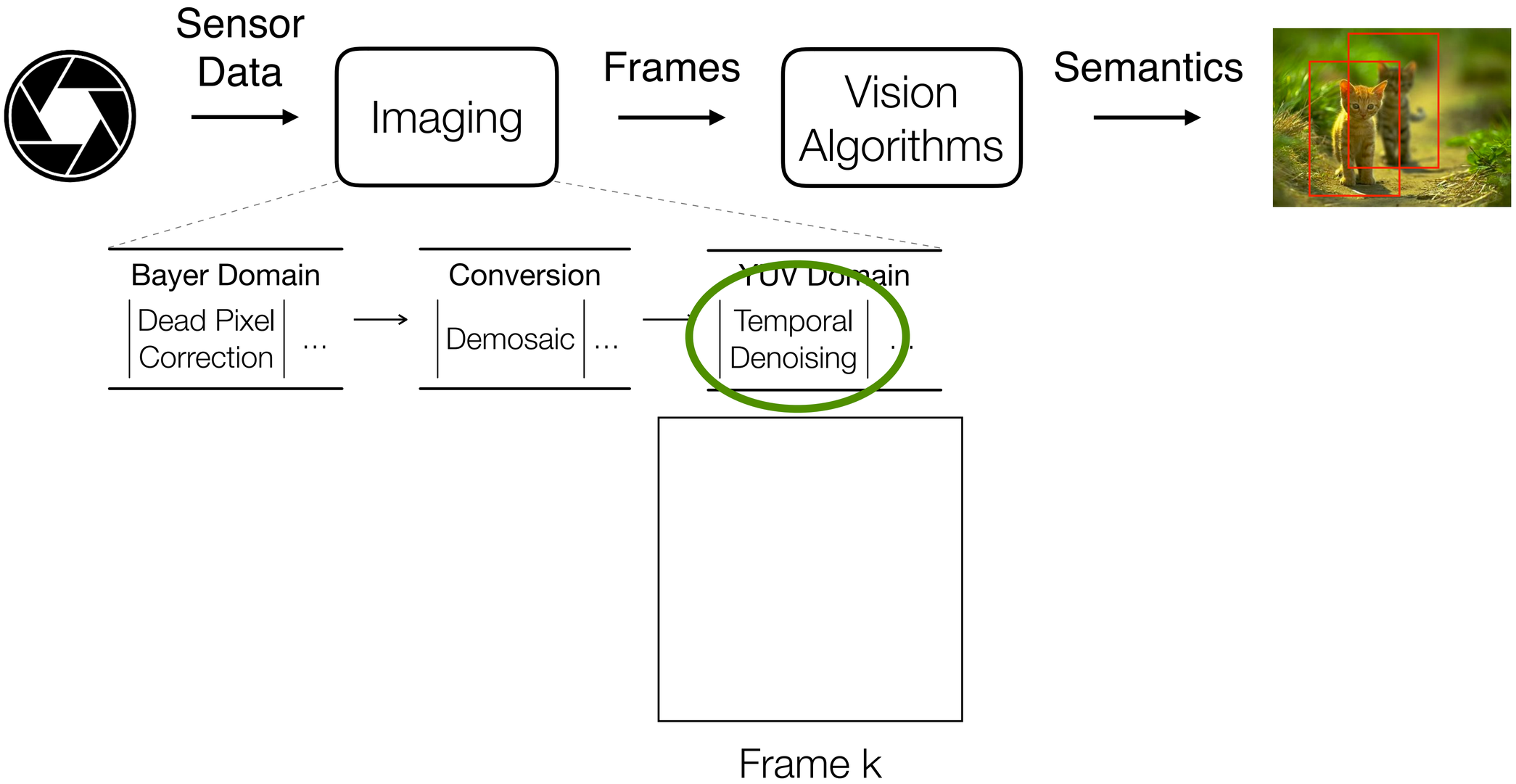
Co-Designing Imaging with Vision



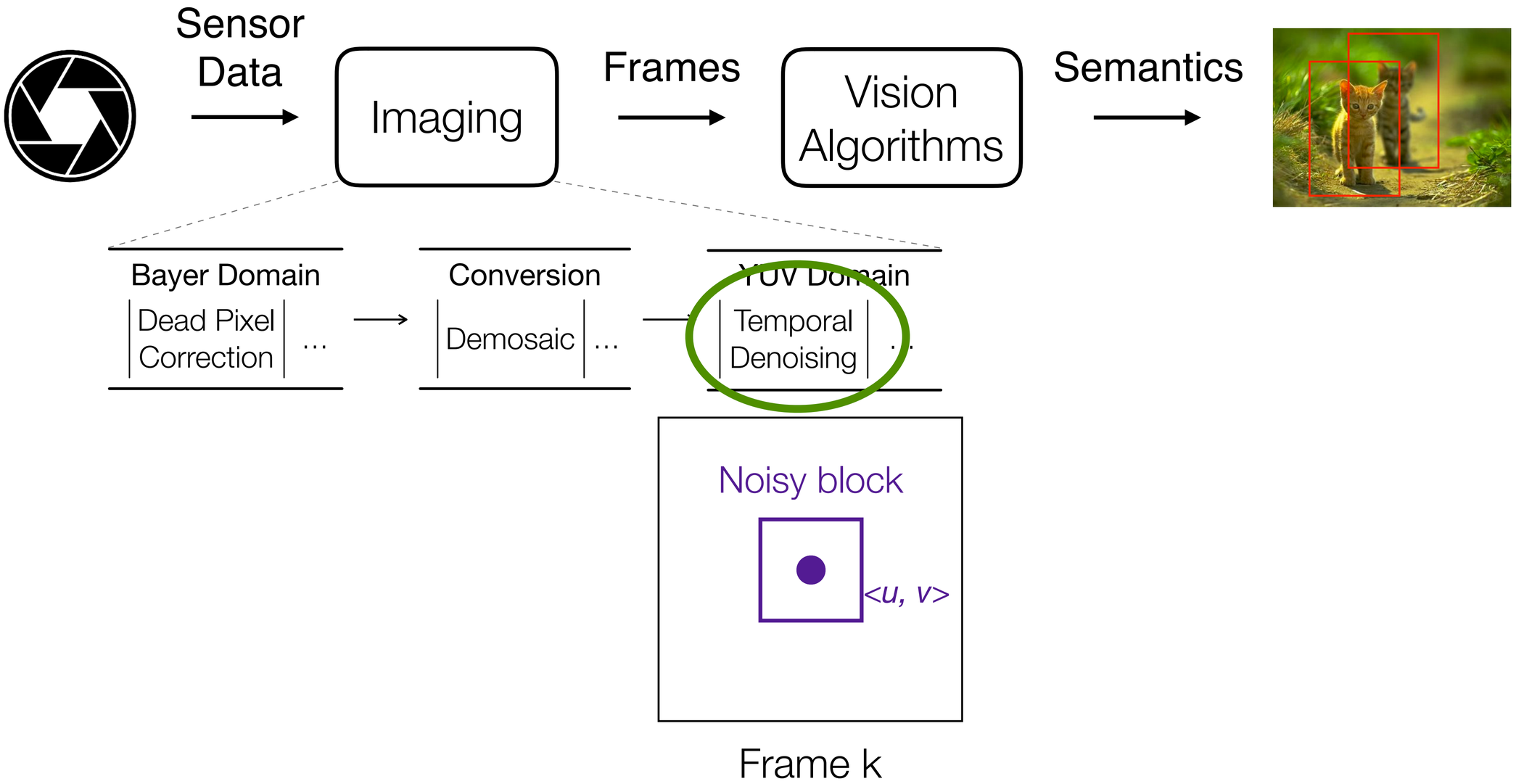
Co-Designing Imaging with Vision



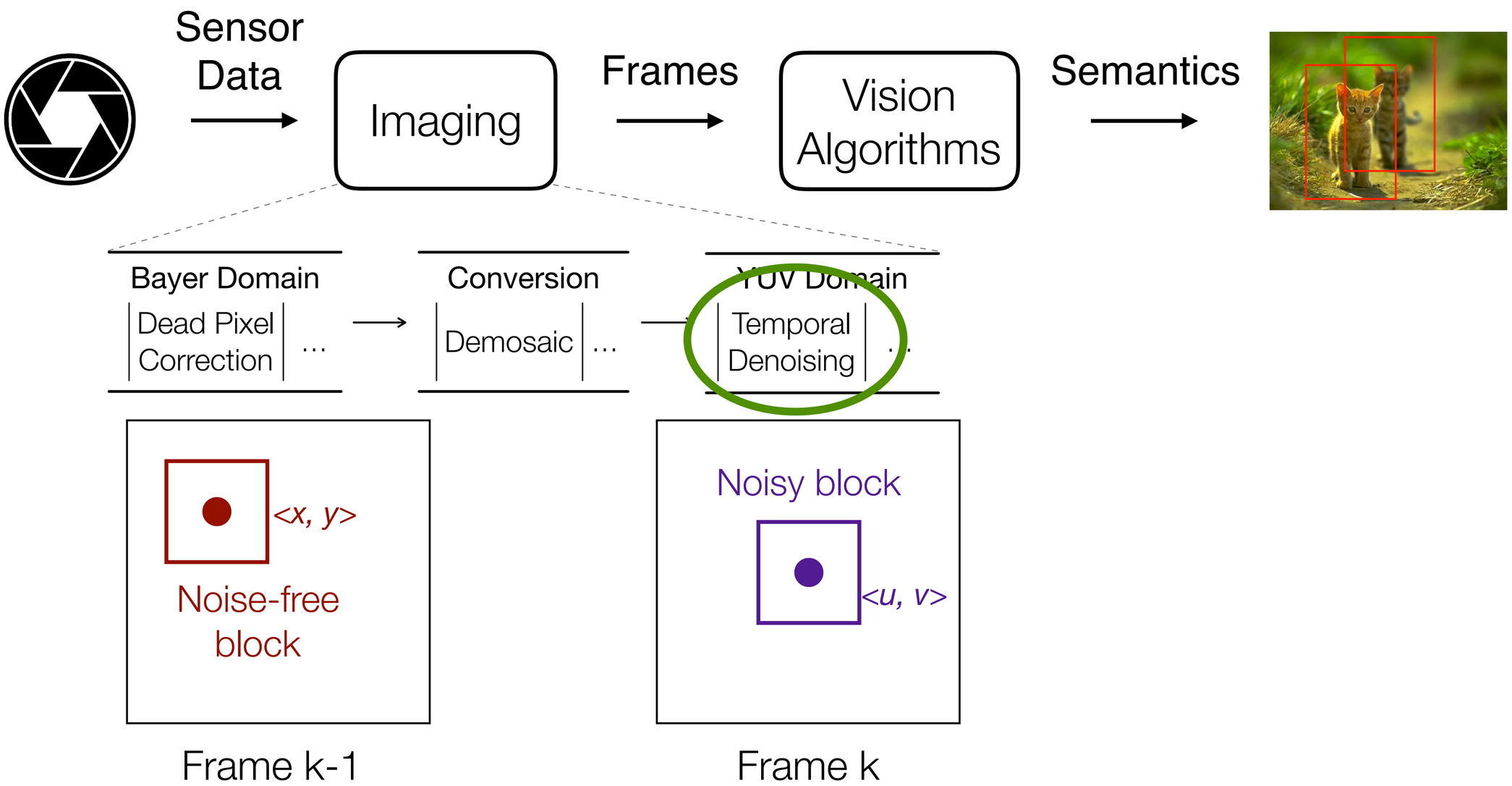
Co-Designing Imaging with Vision



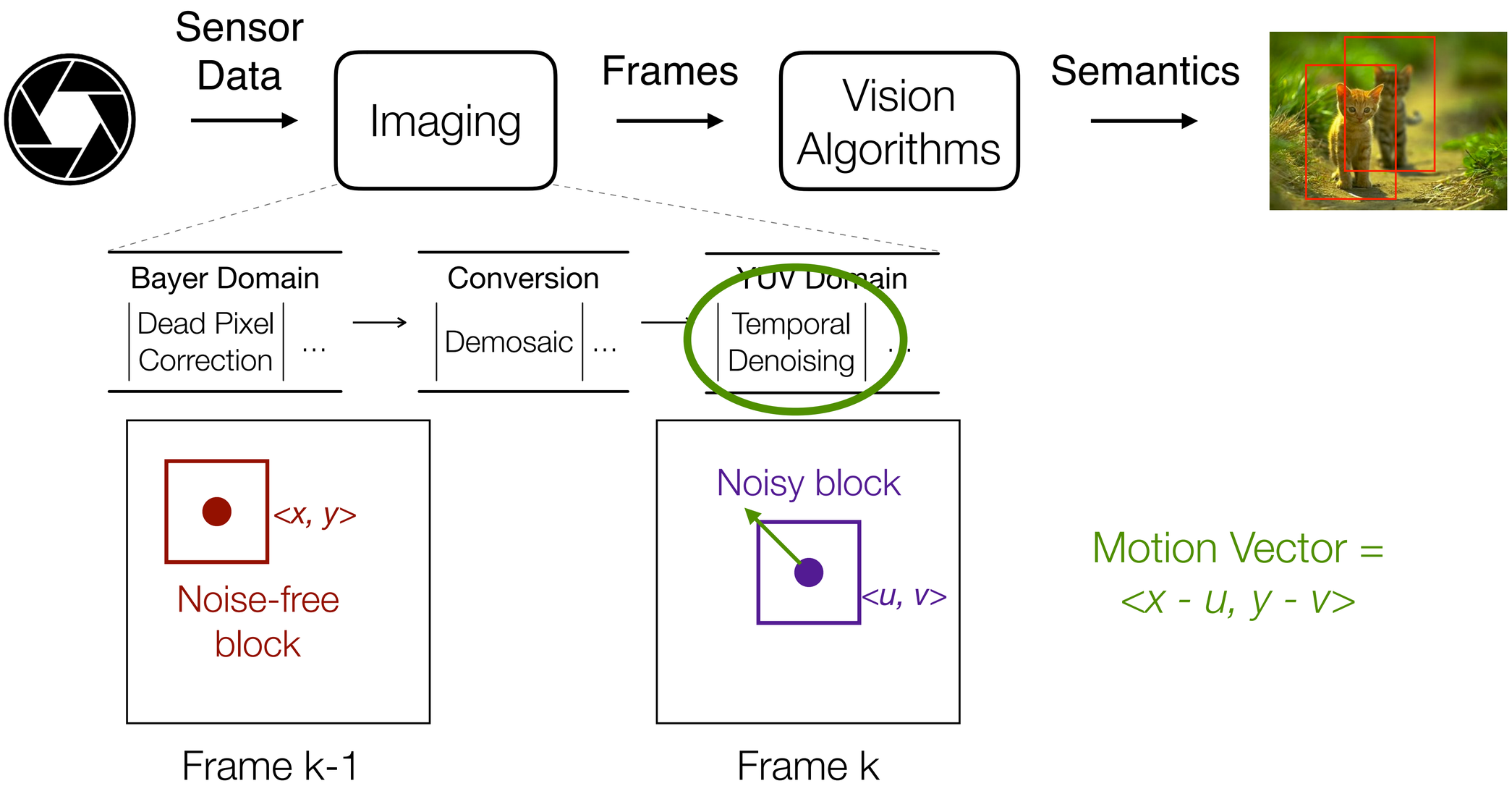
Co-Designing Imaging with Vision



Co-Designing Imaging with Vision

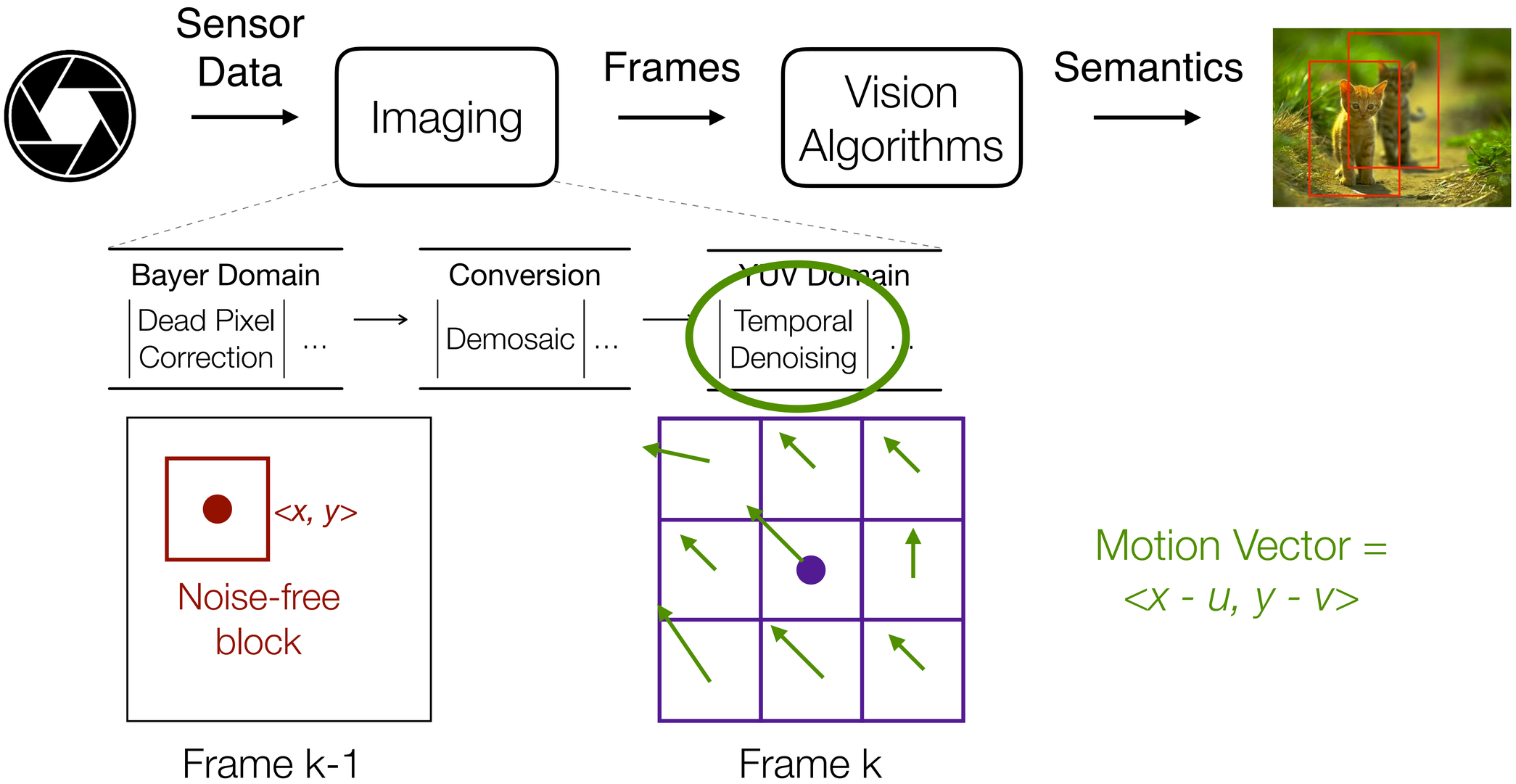


Co-Designing Imaging with Vision

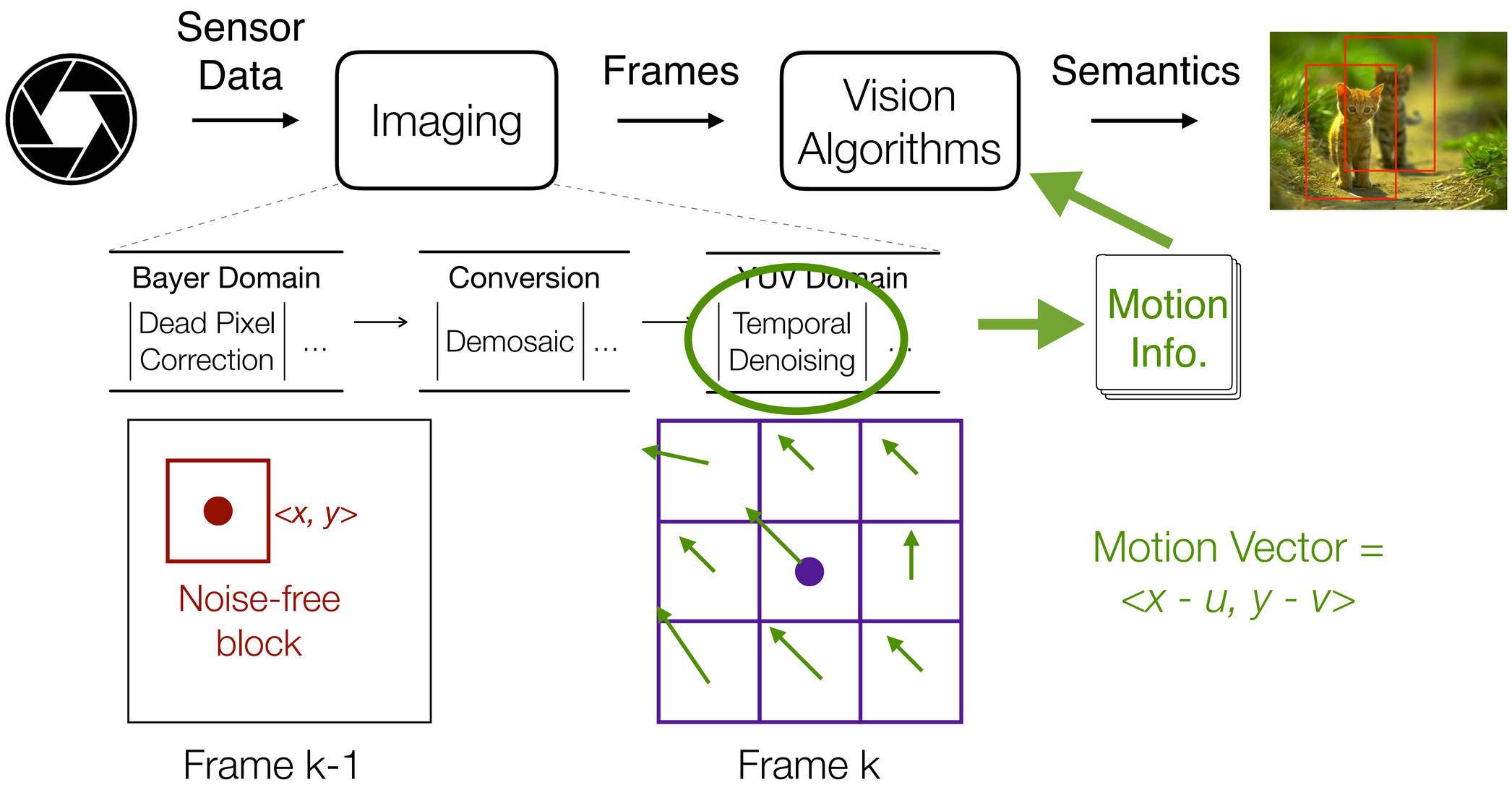


Motion Vector = <x - u, y - v>

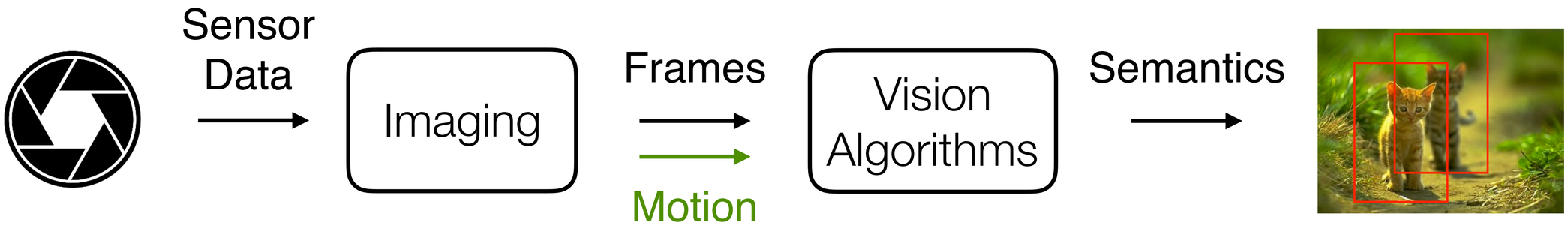
Co-Designing Imaging with Vision



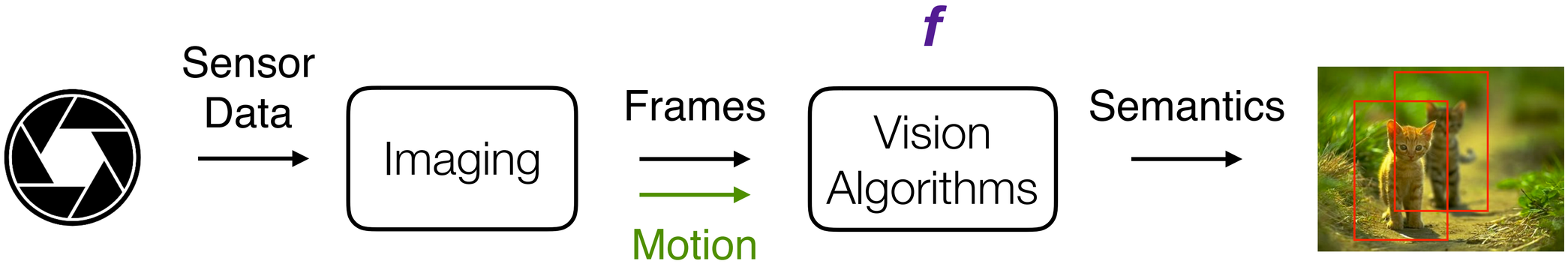
Co-Designing Imaging with Vision



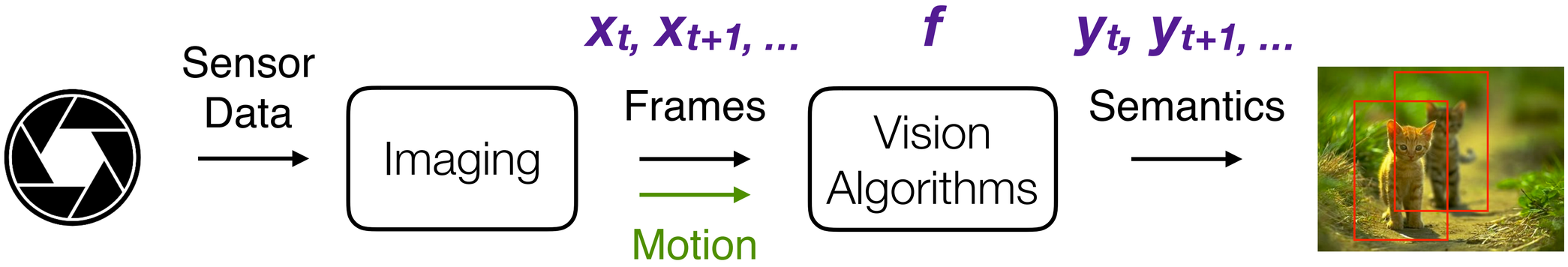
Using Motion for Incremental Visual Computing



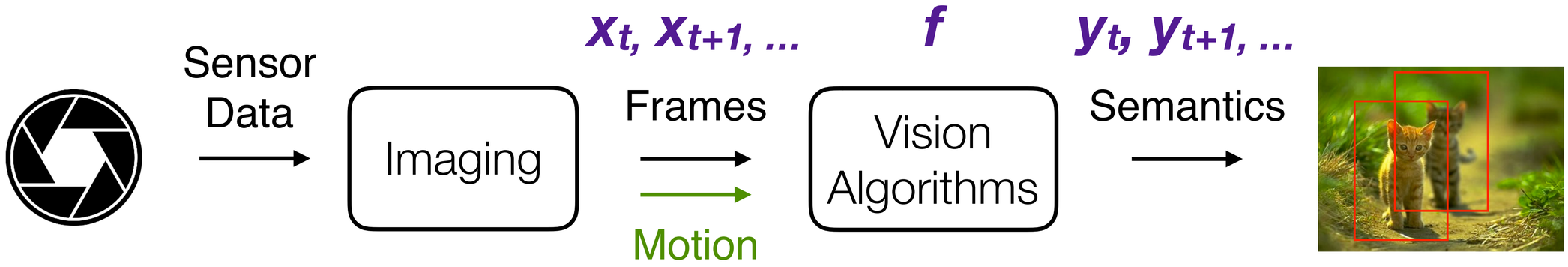
Using Motion for Incremental Visual Computing



Using Motion for Incremental Visual Computing



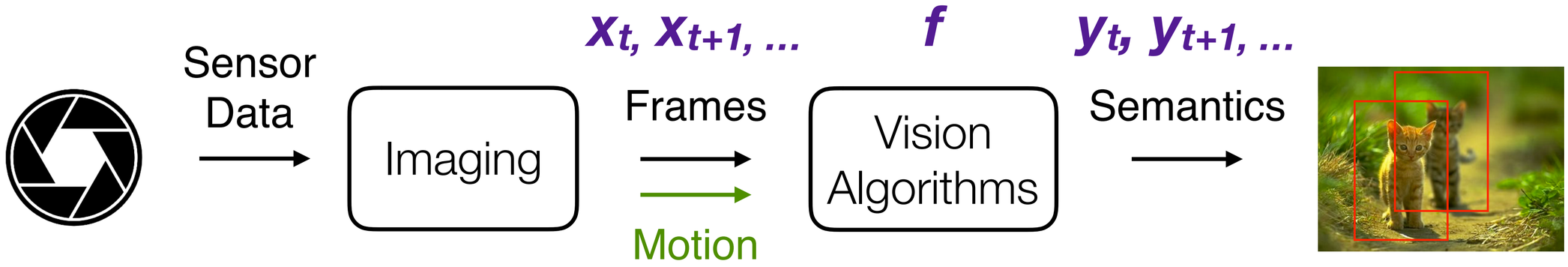
Using Motion for Incremental Visual Computing



$y_t = f(x_t)$

$y_{t+1} = f(x_{t+1})$

Using Motion for Incremental Visual Computing



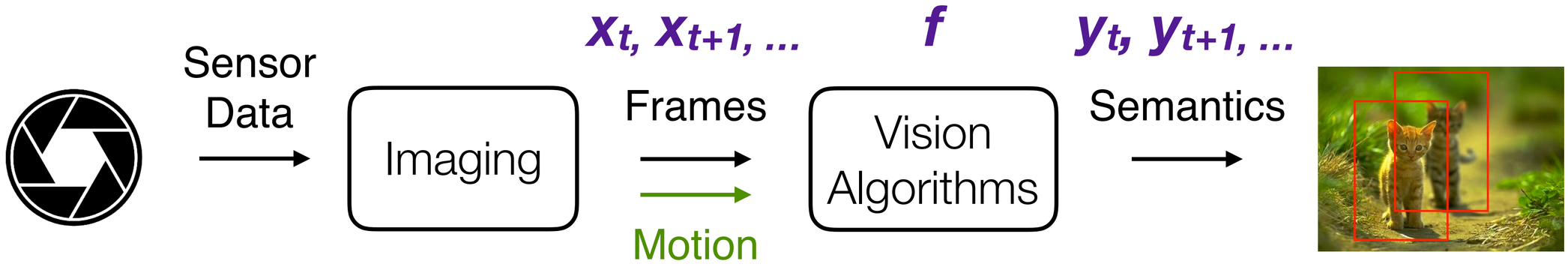
$$y_t = f(x_t)$$

$$y_{t+1} = f(x_{t+1})$$

$$\Delta y = y_{t+1} - y_t$$

$$\Delta x = x_{t+1} - x_t$$

Using Motion for Incremental Visual Computing

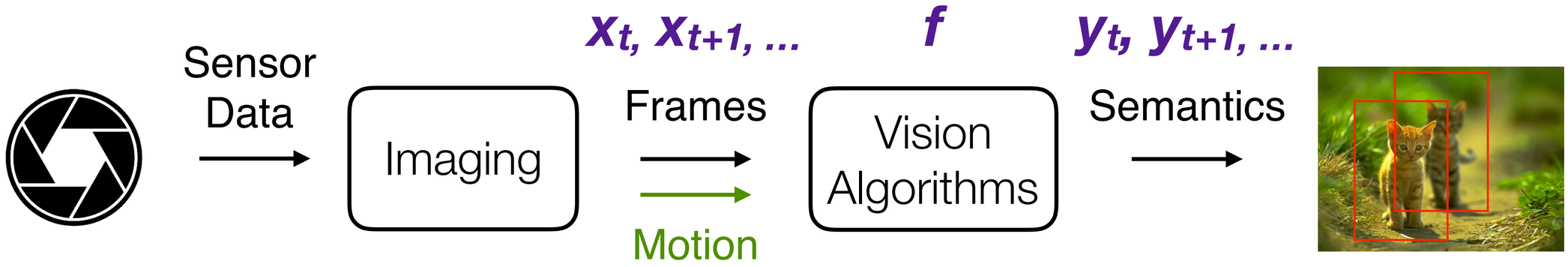


$$y_t = f(x_t)$$

$$y_{t+1} = f(x_{t+1})$$

$$\rightarrow \quad \frac{\Delta y}{\Delta x} = \frac{y_{t+1} - y_t}{x_{t+1} - x_t}$$

Using Motion for Incremental Visual Computing

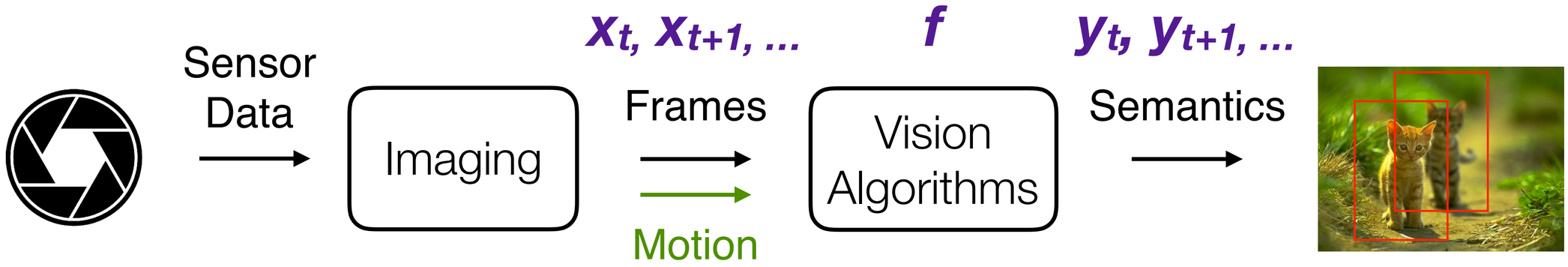


$$y_t = f(x_t)$$

$$y_{t+1} = f(x_{t+1})$$

$$\rightarrow \quad \frac{\Delta y}{\Delta x} = \frac{y_{t+1} - y_t}{x_{t+1} - x_t} \quad \rightarrow \quad y_{t+1} = \Delta y / \Delta x * (x_{t+1} - x_t) + y_t$$

Using Motion for Incremental Visual Computing



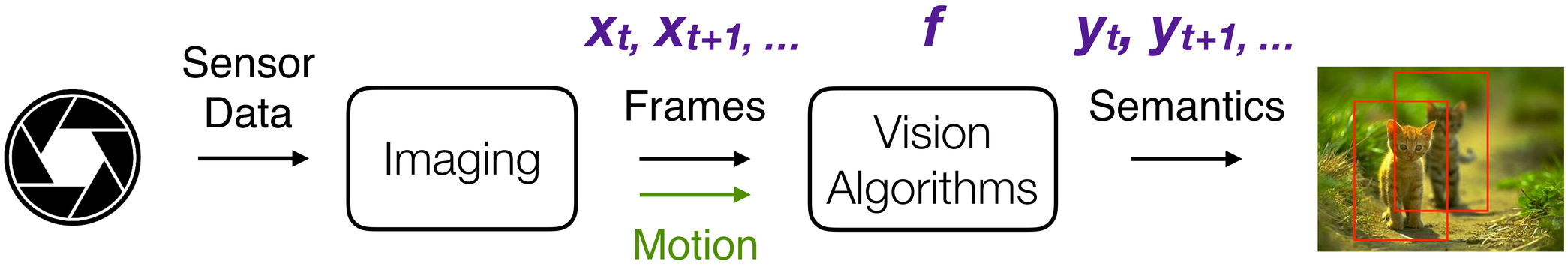
$y_t = f(x_t)$

$\boxed{y_{t+1} = f(x_{t+1})}$

$$\frac{\Delta y}{\Delta x} = \frac{y_{t+1} - y_t}{x_{t+1} - x_t}$$

$$\boxed{y_{t+1} = \Delta y/\Delta x * (x_{t+1} - x_t) + y_t}$$

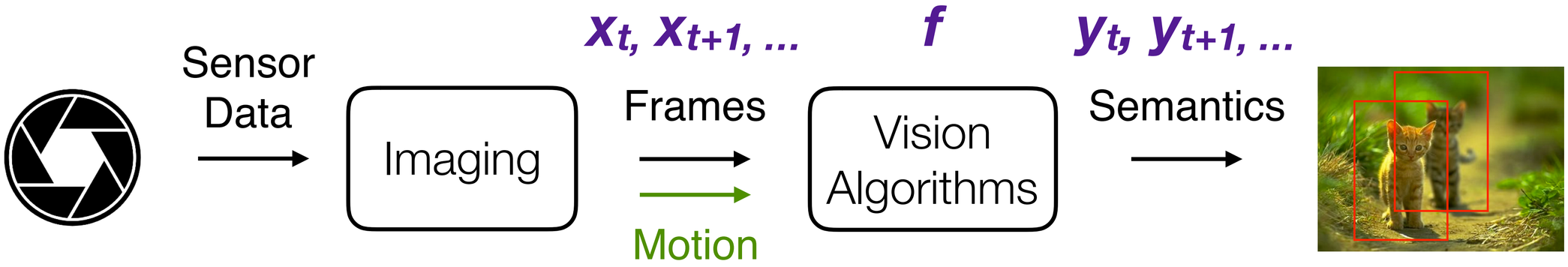
Using Motion for Incremental Visual Computing



Three Key Operators:

$$y_{t+1} = \Delta y/\Delta x * (x_{t+1} - x_t) + y_t$$

Using Motion for Incremental Visual Computing

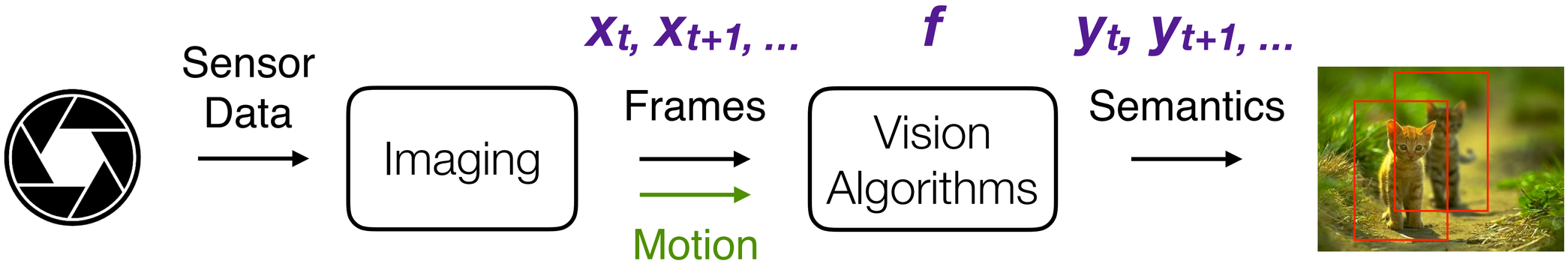


Three Key Operators:

- Calculate input diff. ⊖

$$y_{t+1} = \Delta y/\Delta x * \boxed{(x_{t+1} - x_t)} + y_t$$

Using Motion for Incremental Visual Computing

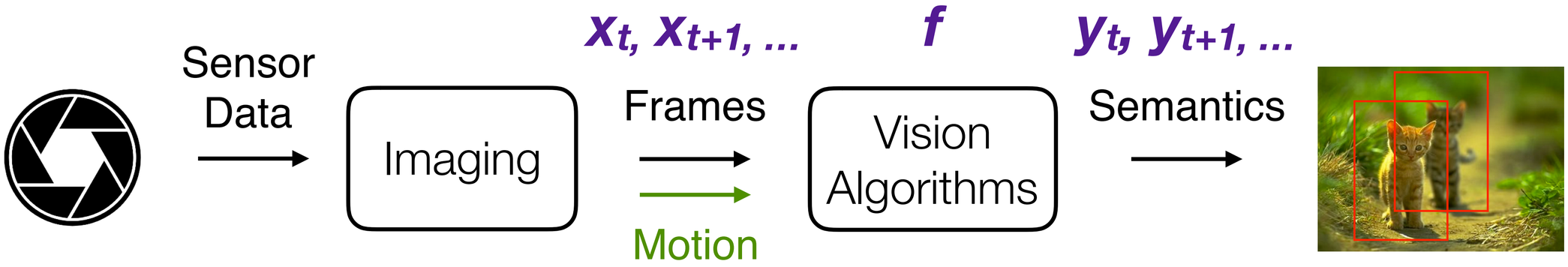


Three Key Operators:

- Calculate input diff. ⊖
- Convert input diff. to output diff. ***df***

$$y_{t+1} = \boxed{\Delta y/\Delta x\ *}(x_{t+1} - x_t) + y_t$$

Using Motion for Incremental Visual Computing

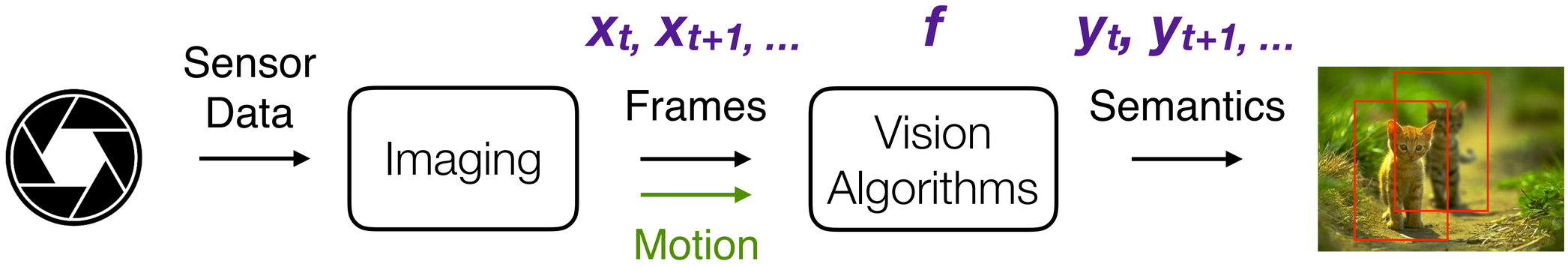


Three Key Operators:

- Calculate input diff. ⊖
- Convert input diff. to output diff. ***df***
- Update output ⊕

$$y_{t+1} = \Delta y/\Delta x * (x_{t+1} - x_t) \boxed{+ y_t}$$

Using Motion for Incremental Visual Computing



Three Key Operators:

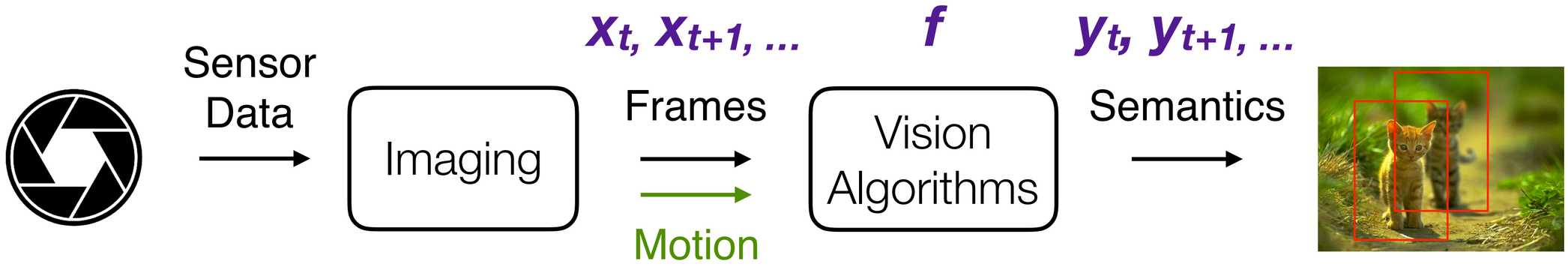
- Calculate input diff. $\ominus$
- Convert input diff. to output diff. ***df***
- Update output $\oplus$

$$y_{t+1} = \Delta y/\Delta x * (x_{t+1} - x_t) + y_t$$

$$\downarrow$$

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$

Using Motion for Incremental Visual Computing



Three Key Operators:

- Calculate input diff. $\ominus$
- Convert input diff. to output diff. ***df***
- Update output $\oplus$

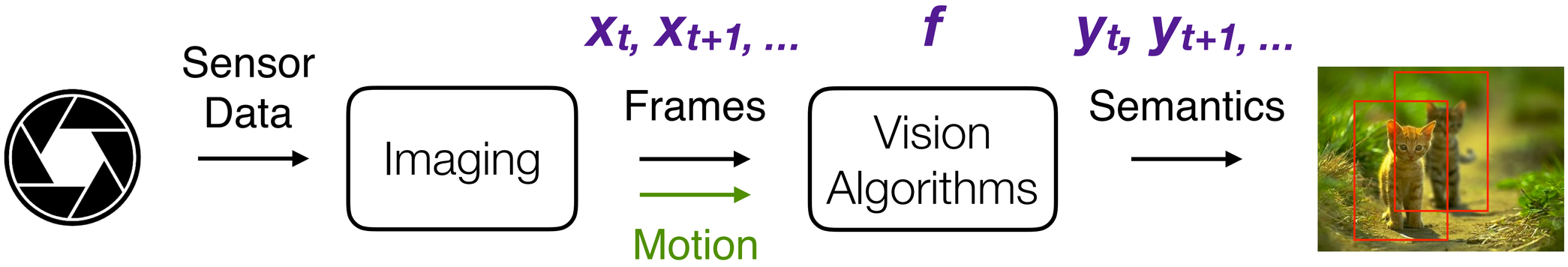
$$y_{t+1} = \Delta y/\Delta x * (x_{t+1} - x_t) + y_t$$

$$\downarrow$$

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$

$$= f(x_{t+1})$$

Using Motion for Incremental Visual Computing



Three Key Operators:

- Calculate input diff. $\ominus$
- Convert input diff. to output diff. ***df***
- Update output $\oplus$

Expectation:

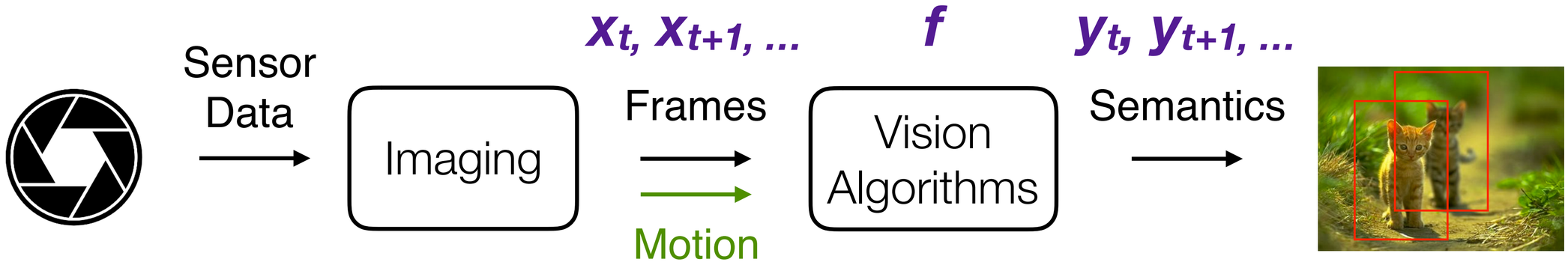
- ($\ominus$ + ***df*** + $\oplus$) more efficient than ***f***

$$y_{t+1} = \Delta y/\Delta x * (x_{t+1} - x_t) + y_t$$

$$\downarrow$$

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$
$$= f(x_{t+1})$$

Using Motion for Incremental Visual Computing



Free!

Three Key Operators:

- Calculate input diff. $\ominus$
- Convert input diff. to output diff. ***df***
- Update output $\oplus$

Expectation:

- ($\ominus$ + ***df*** + $\oplus$) more efficient than ***f***

$$y_{t+1} = \Delta y/\Delta x * (x_{t+1} - x_t) + y_t$$

$$\downarrow$$

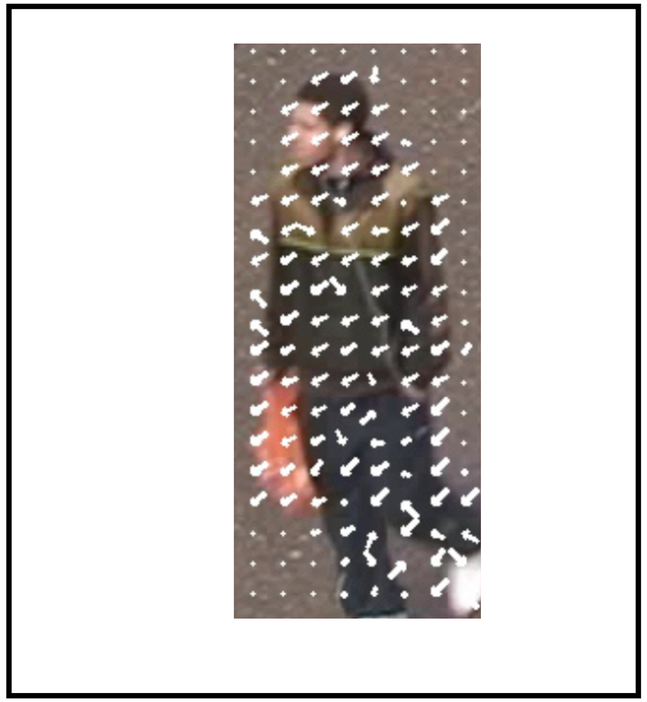
$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$
$$= f(x_{t+1})$$

A (Trivial) Monocular Example: Object Tracking

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$

A (Trivial) Monocular Example: Object Tracking

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$



A (Trivial) Monocular Example: Object Tracking

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$

Calculate input difference: $\ominus$

- Motion vector (expensive, but obtained from imaging “for free”)

A (Trivial) Monocular Example: Object Tracking

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$

Calculate input difference: $\ominus$

- Motion vector (expensive, but obtained from imaging “for free”)

Convert input diff. to output diff.: ***df***

- Averaging motion vectors within the Region of Interest

A (Trivial) Monocular Example: Object Tracking

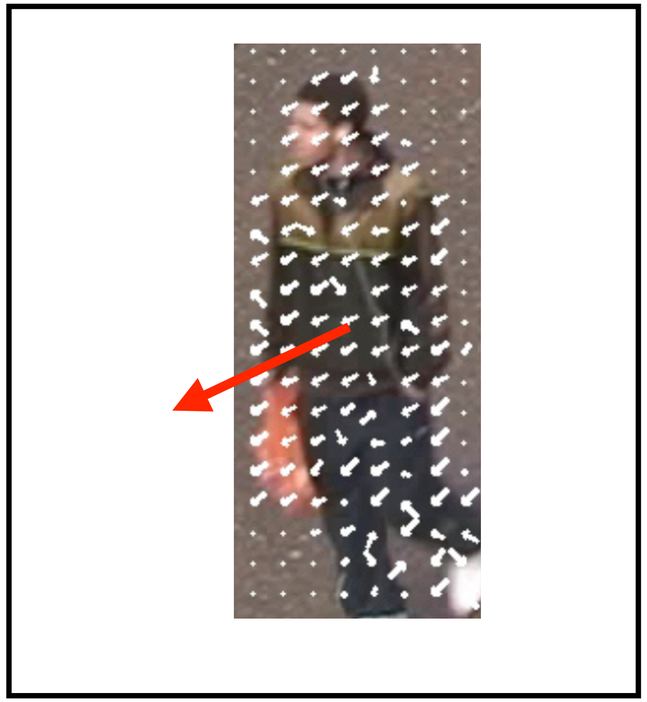
$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$

Calculate input difference: $\ominus$

- Motion vector (expensive, but obtained from imaging “for free”)

Convert input diff. to output diff.: ***df***

- Averaging motion vectors within the Region of Interest



A (Trivial) Monocular Example: Object Tracking

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$

Calculate input difference: $\ominus$

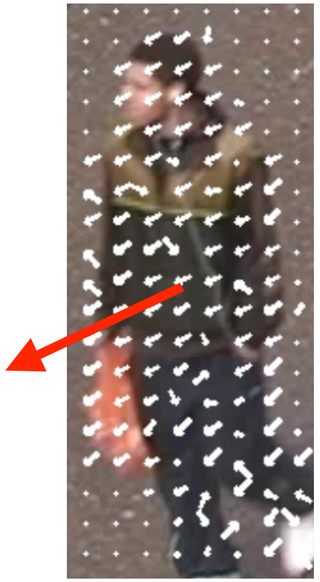
- Motion vector (expensive, but obtained from imaging “for free”)

Convert input diff. to output diff.: ***df***

- Averaging motion vectors within the Region of Interest

Update output: $\oplus$

- Extrapolation



A (Trivial) Monocular Example: Object Tracking

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$

Calculate input difference: $\ominus$

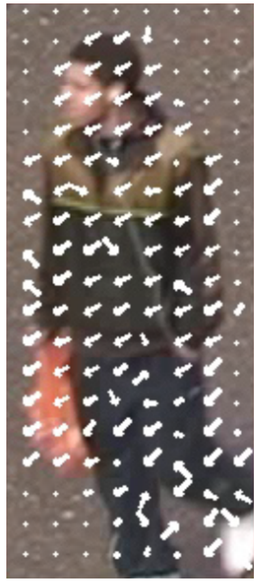
- Motion vector (expensive, but obtained from imaging “for free”)

Convert input diff. to output diff.: ***df***

- Averaging motion vectors within the Region of Interest

Update output: $\oplus$

- Extrapolation



A (Trivial) Monocular Example: Object Tracking

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$

Calculate input difference: $\ominus$

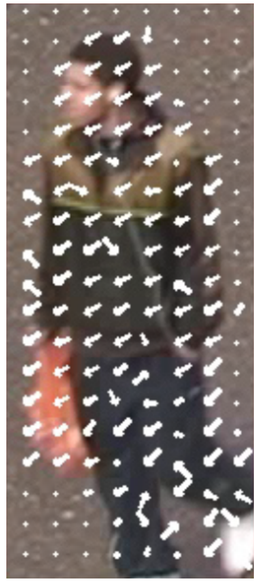
- Motion vector (expensive, but obtained from imaging "for free")

Convert input diff. to output diff.: ***df***

- Averaging motion vectors within the Region of Interest

Update output: $\oplus$

- Extrapolation

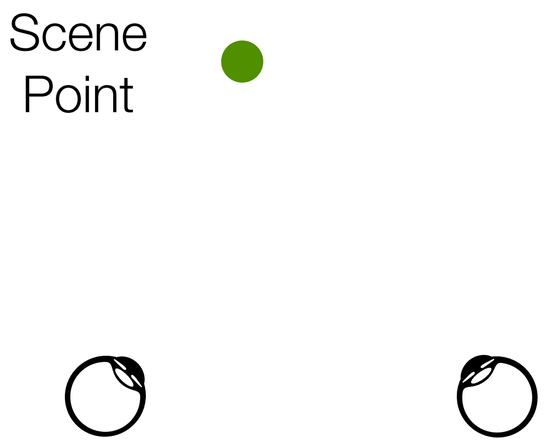


Computationally efficient:

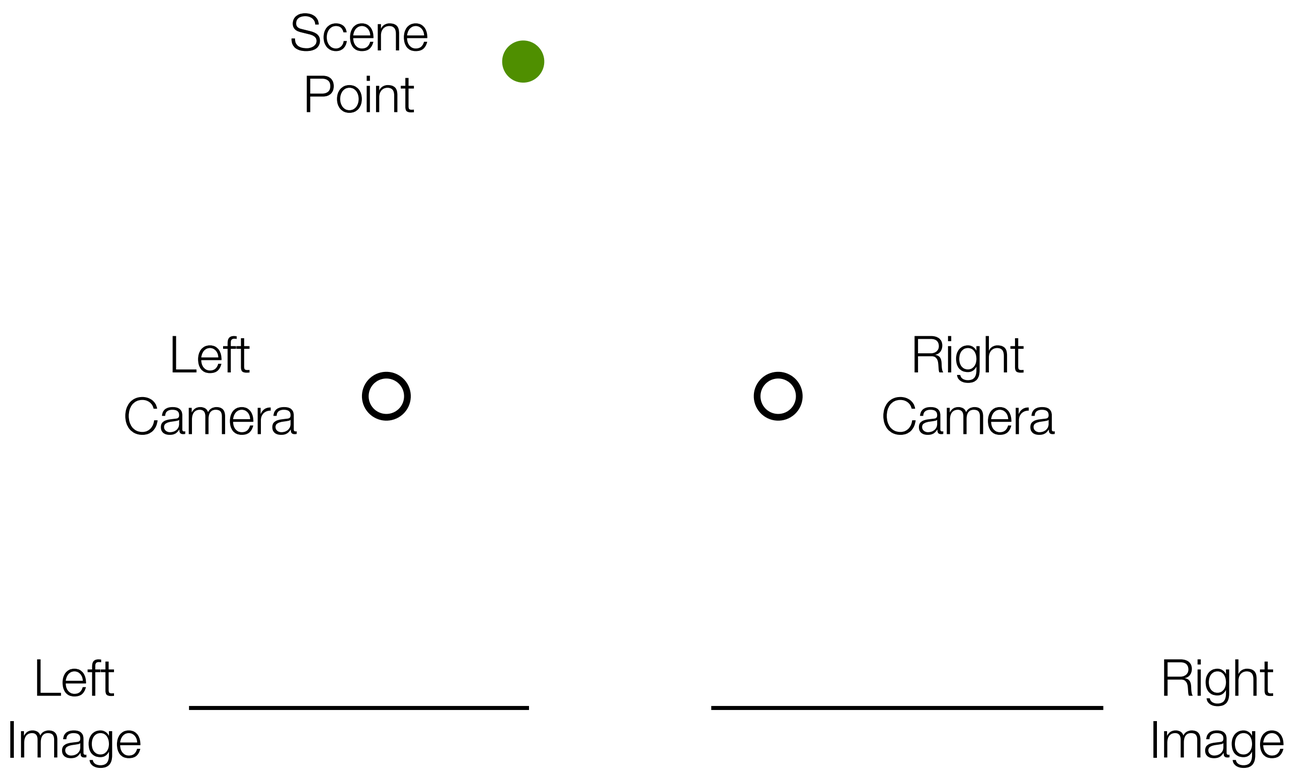
Compute Incrementally: **10K** operations/frame
CNN Inference: **50B** operations/frame

A Binocular Example: Depth Sensing

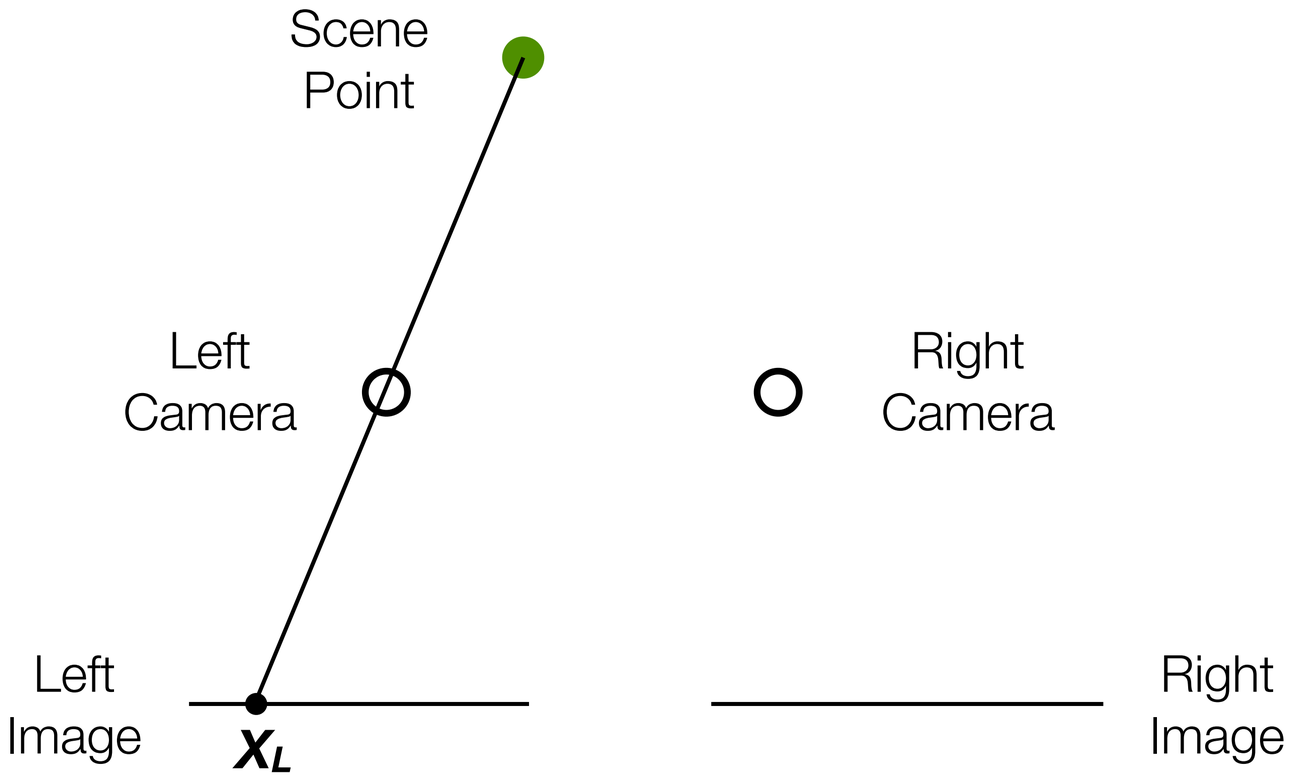
A Binocular Example: Depth Sensing



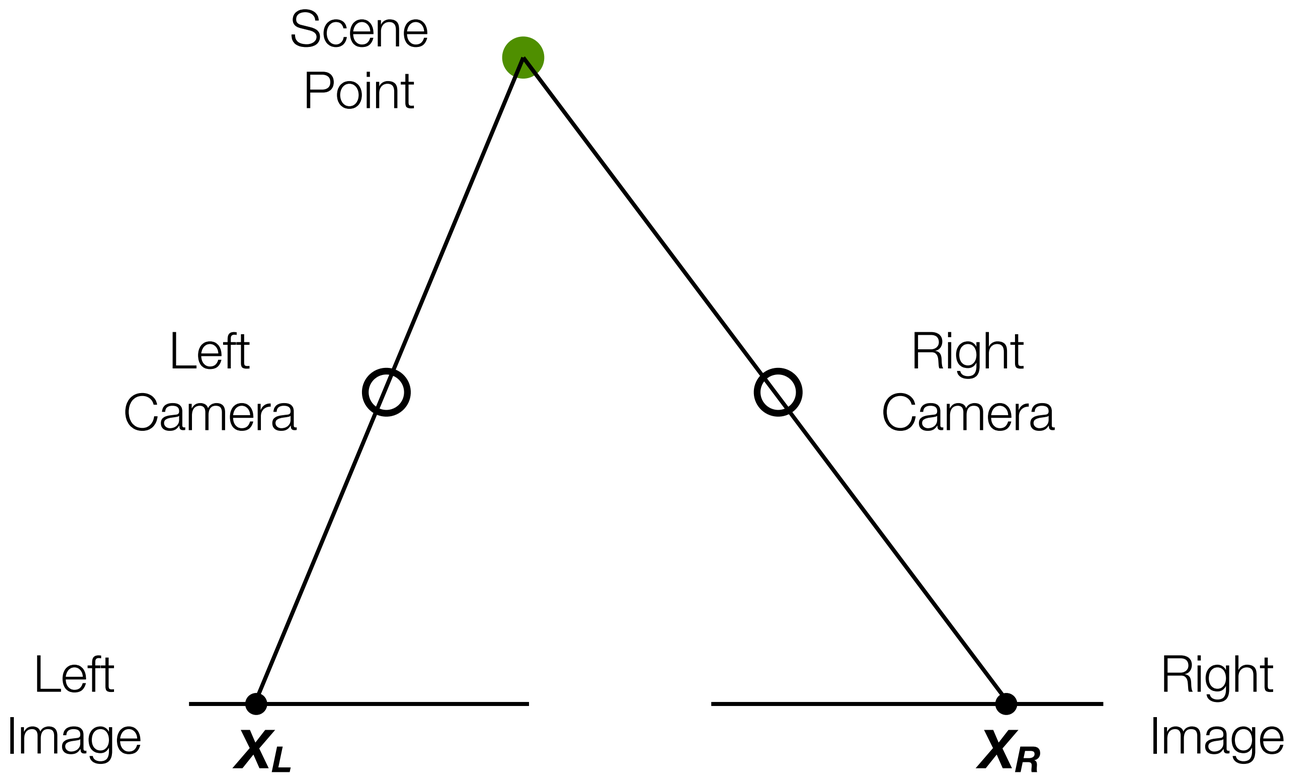
A Binocular Example: Depth Sensing



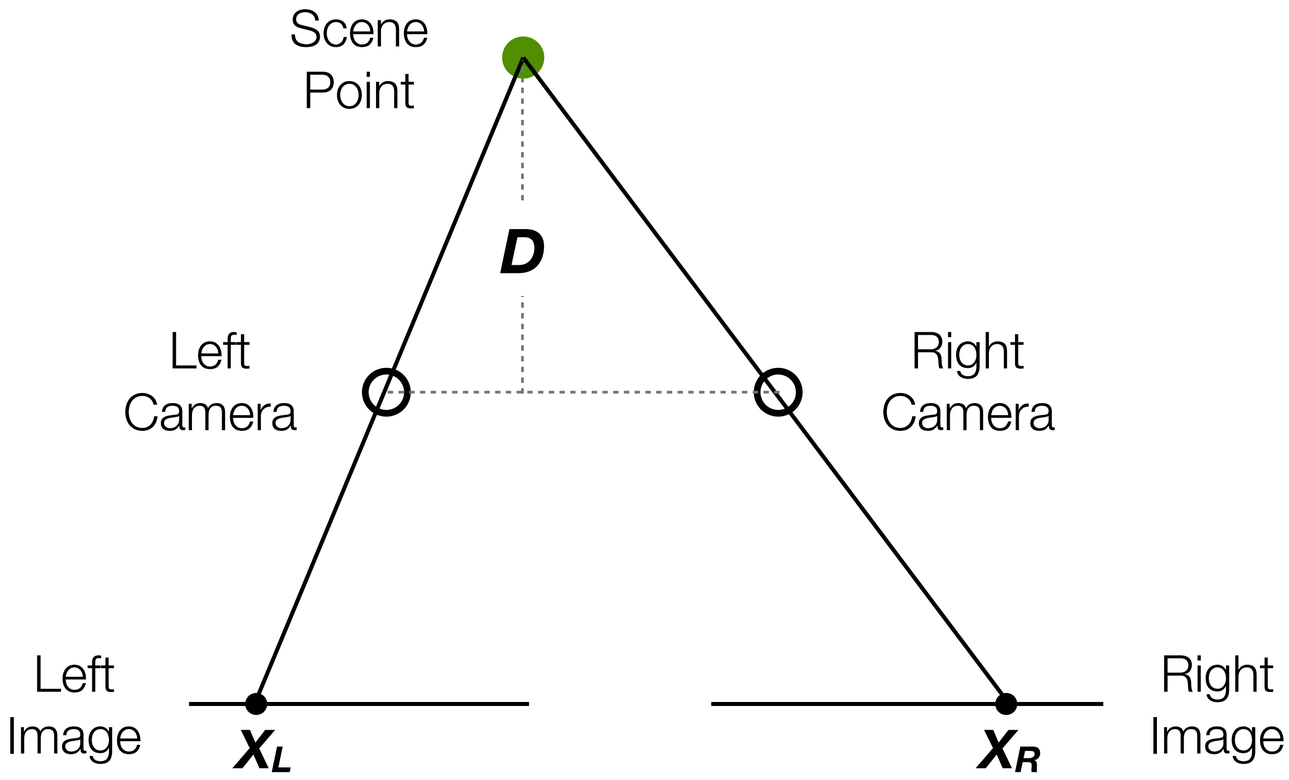
A Binocular Example: Depth Sensing



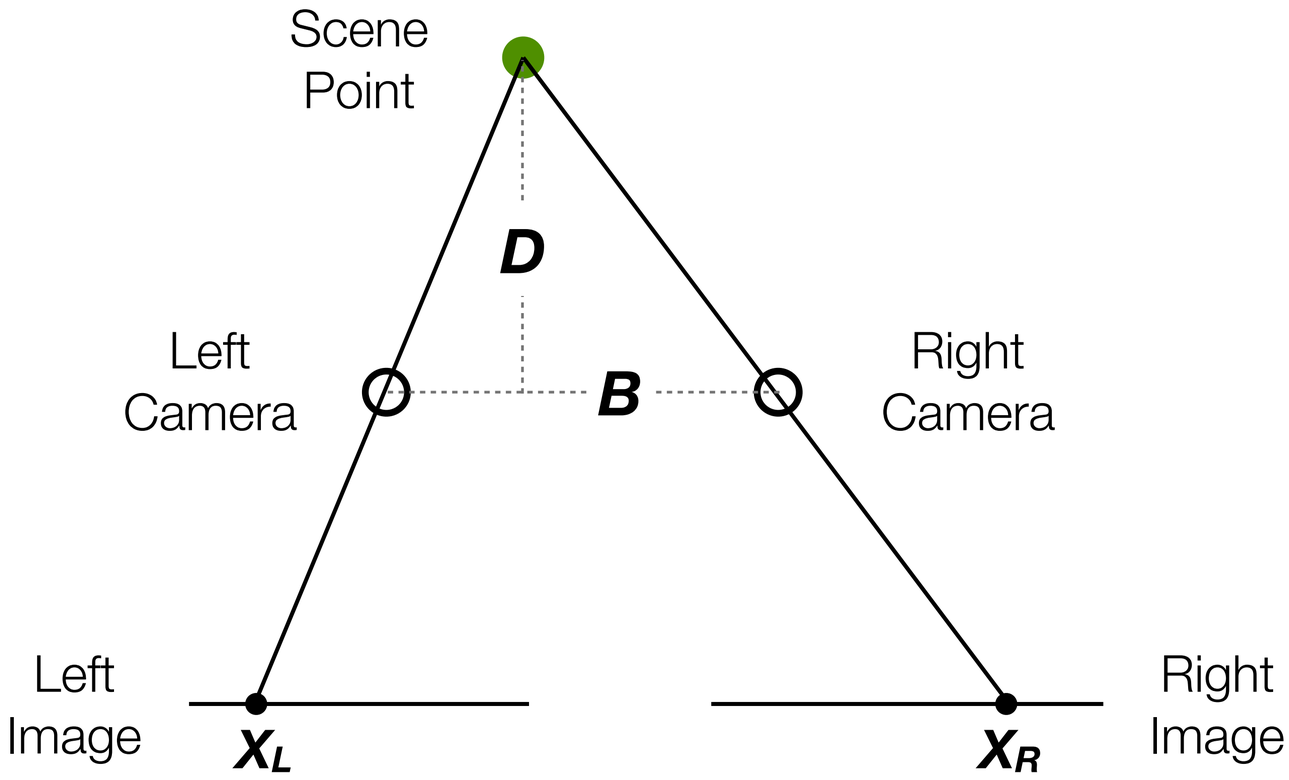
A Binocular Example: Depth Sensing



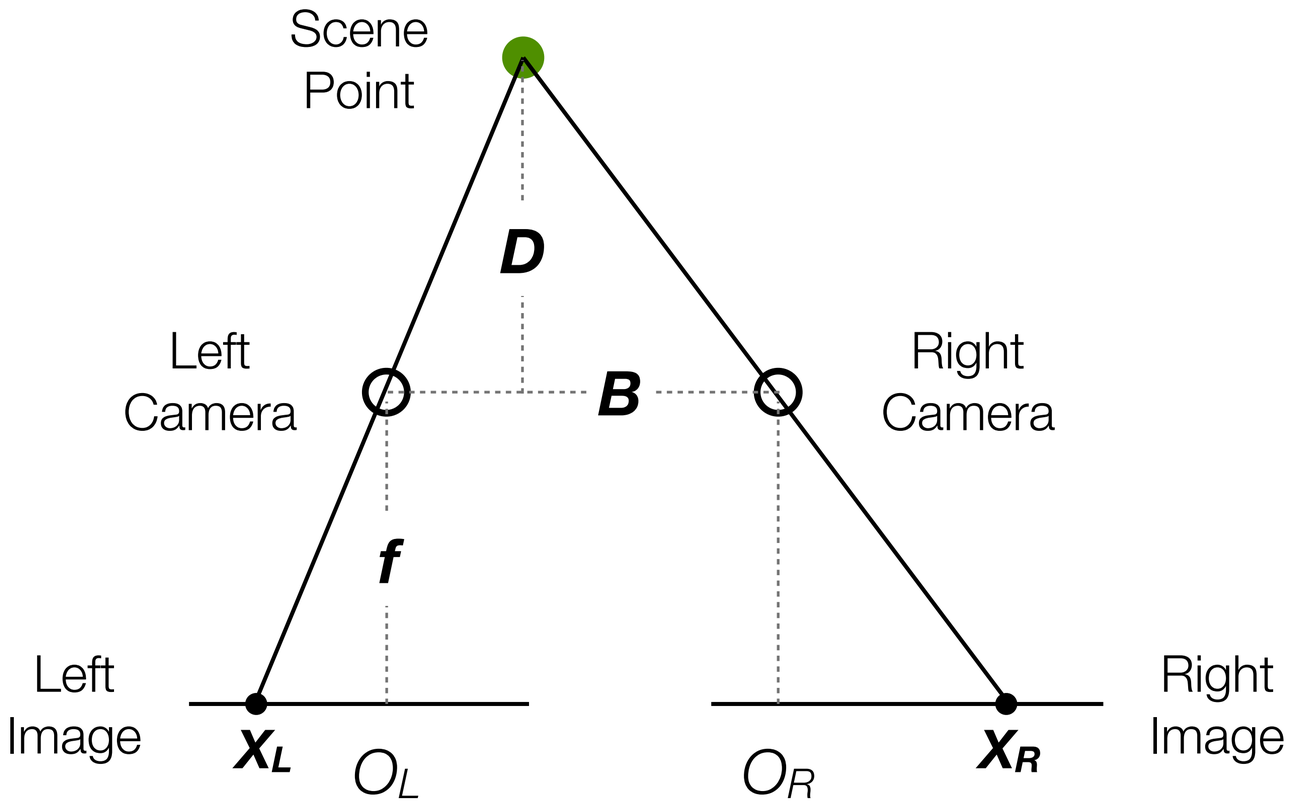
A Binocular Example: Depth Sensing



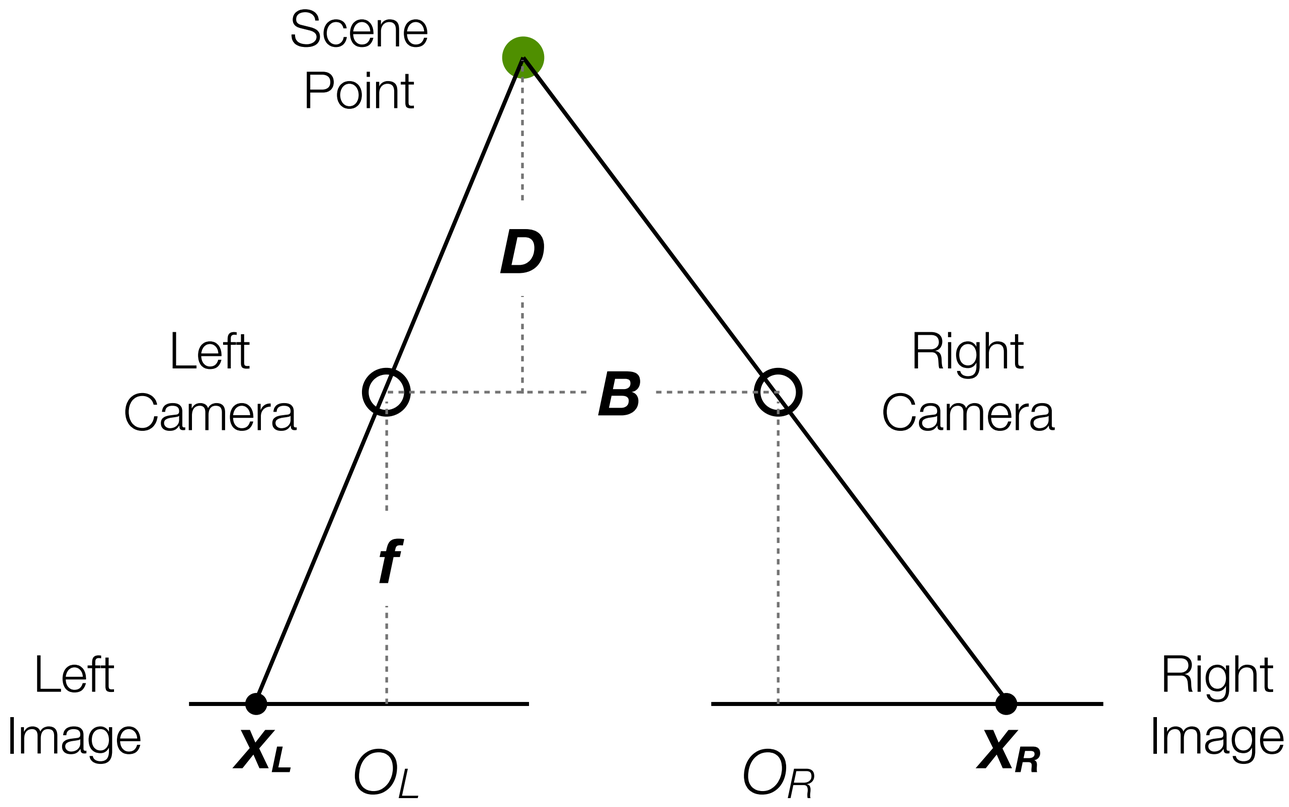
A Binocular Example: Depth Sensing



A Binocular Example: Depth Sensing



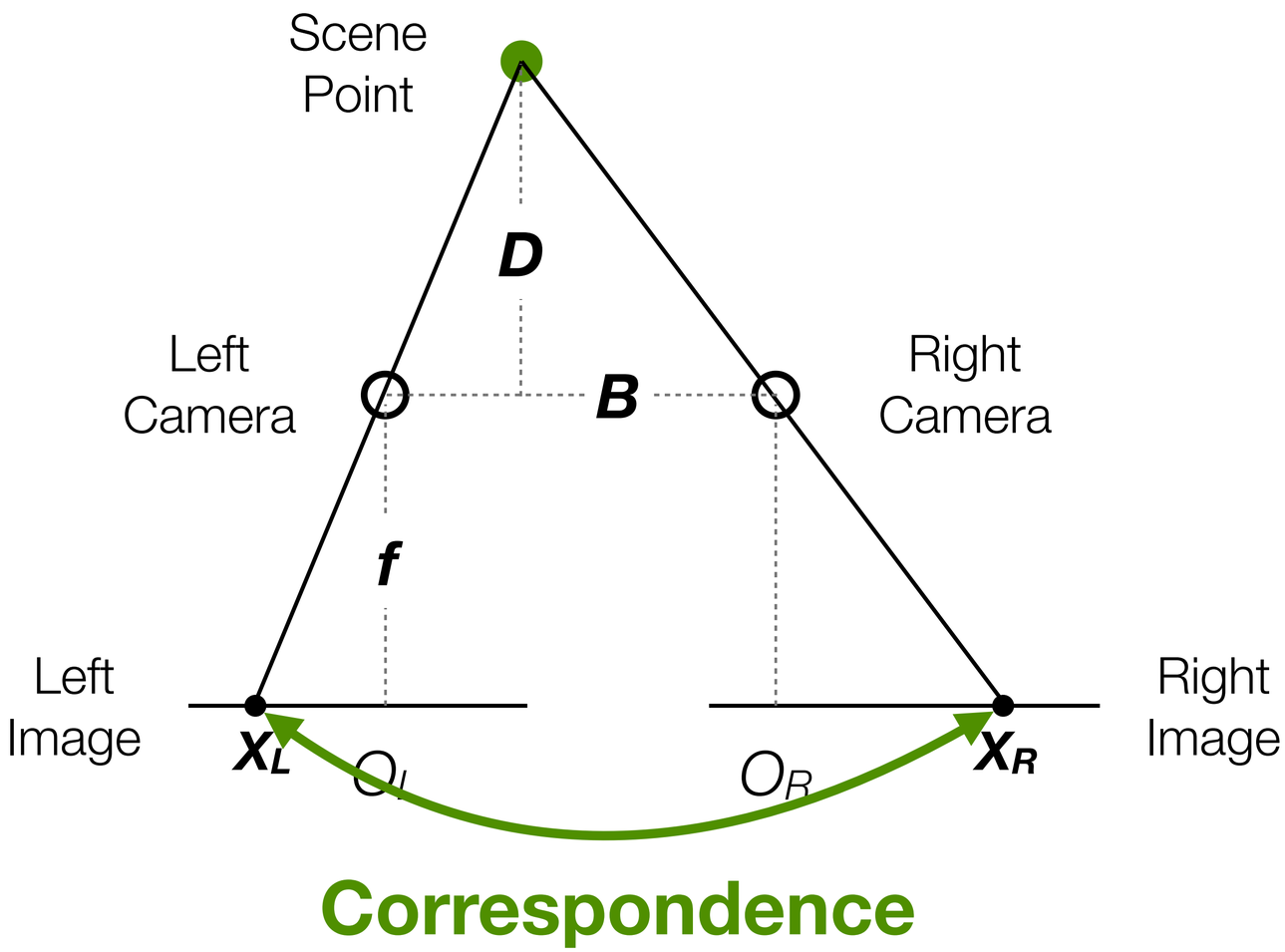
A Binocular Example: Depth Sensing



Using Similar Triangles (Triangulation):

$$D = Bf / (X_R - X_L)$$

A Binocular Example: Depth Sensing



Using Similar Triangles (Triangulation):

$$D = Bf / (X_R - X_L)$$

Disparity

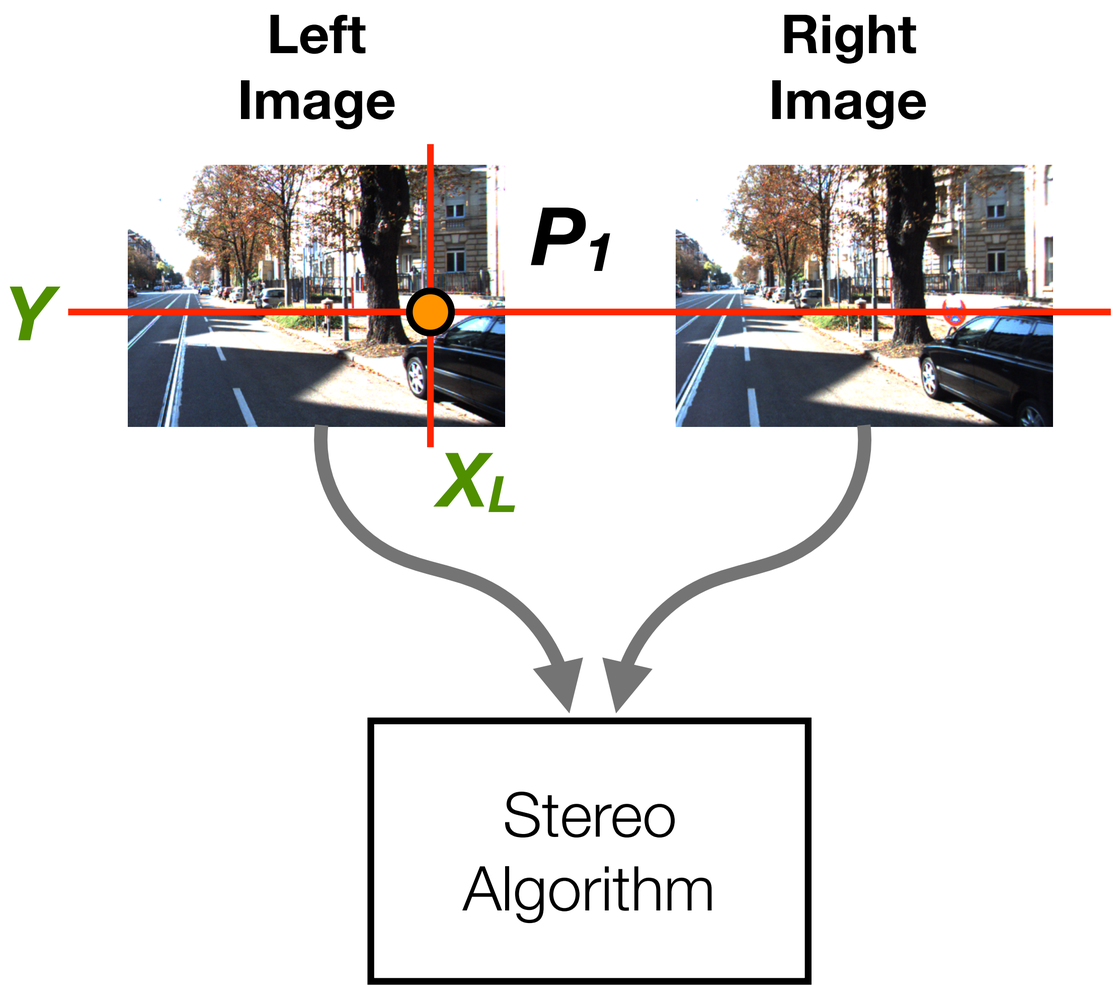
A Binocular Example: Depth Sensing

Left Image

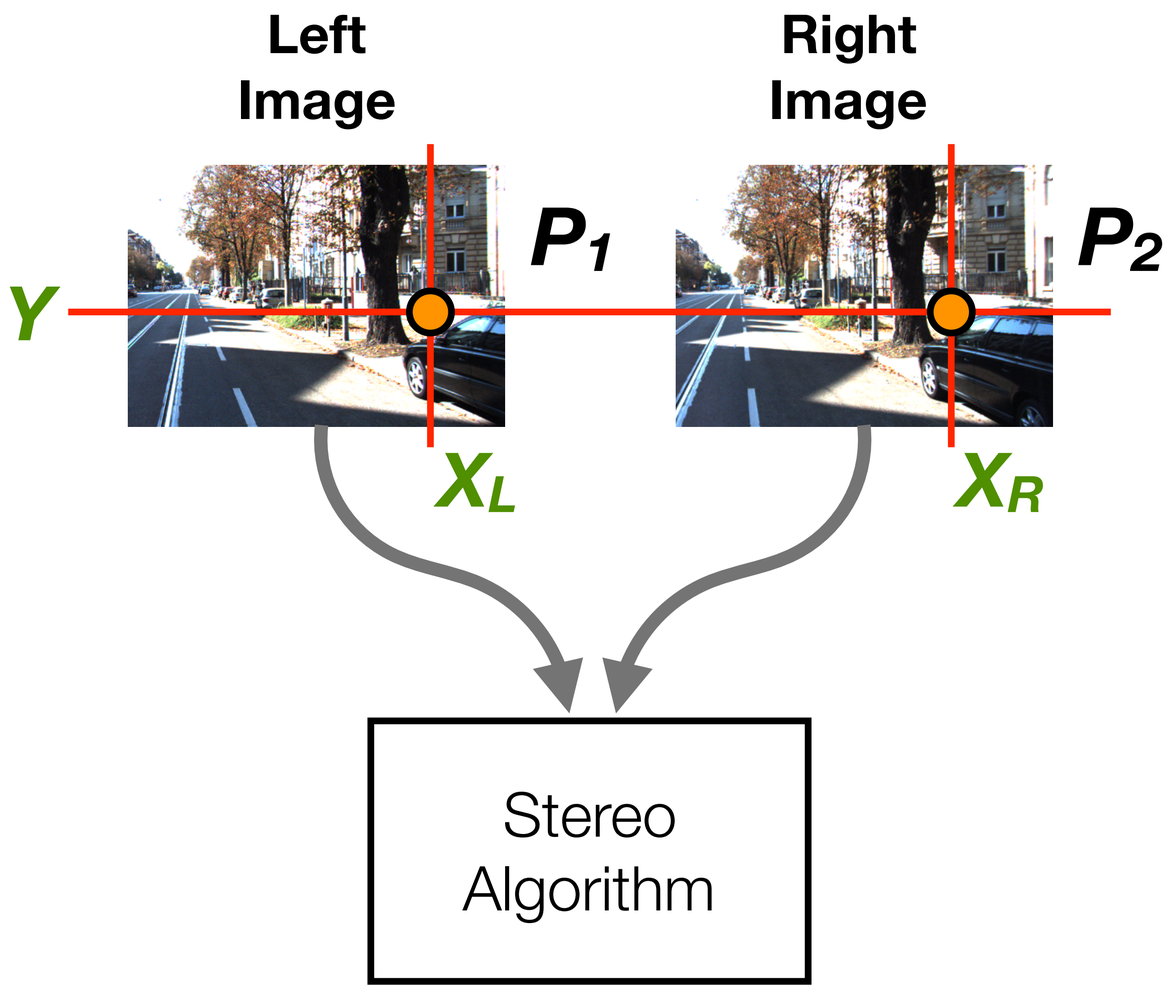
Right Image

Stereo Algorithm

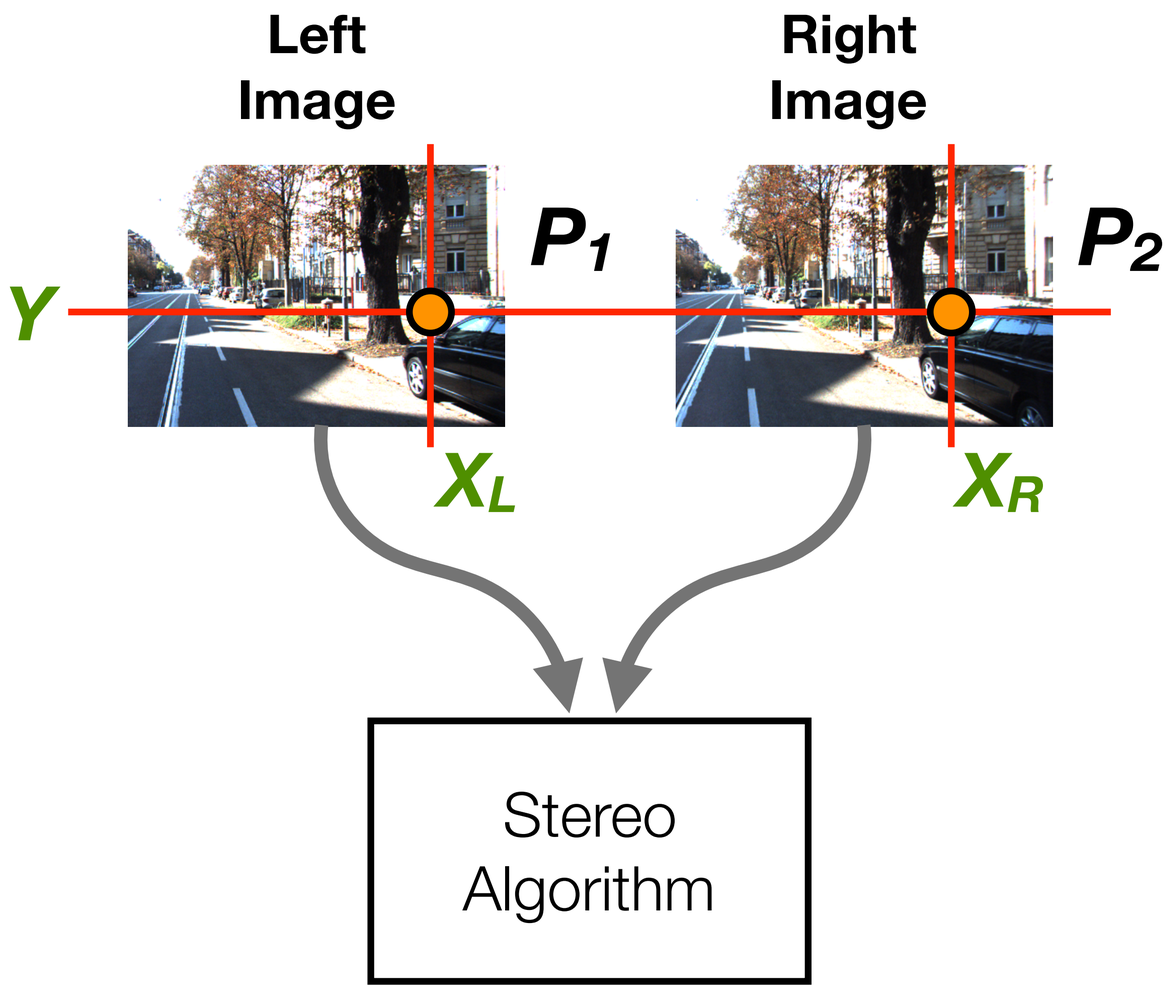
A Binocular Example: Depth Sensing



A Binocular Example: Depth Sensing

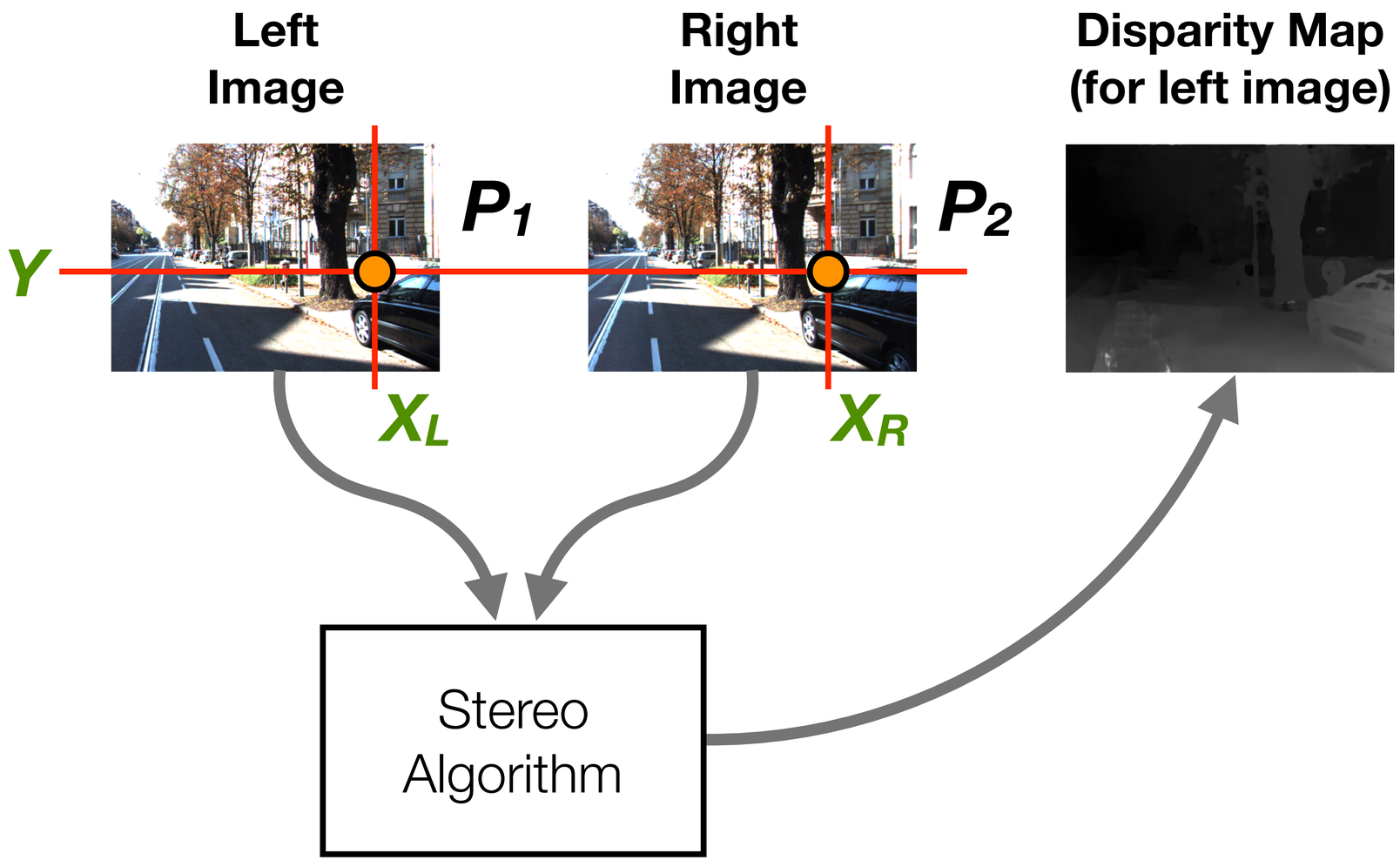


A Binocular Example: Depth Sensing



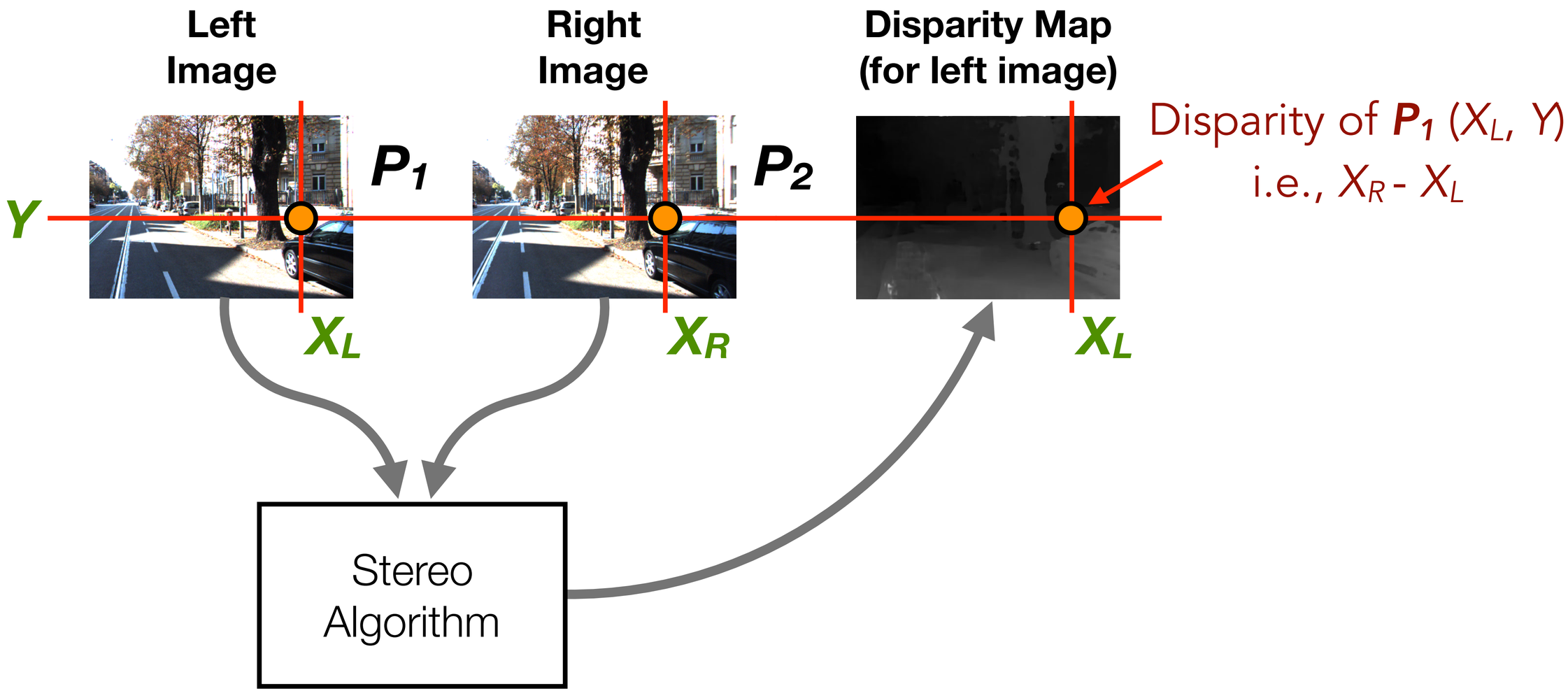
Disparity of P_1 (X_L, Y)
i.e., $X_R - X_L$

A Binocular Example: Depth Sensing

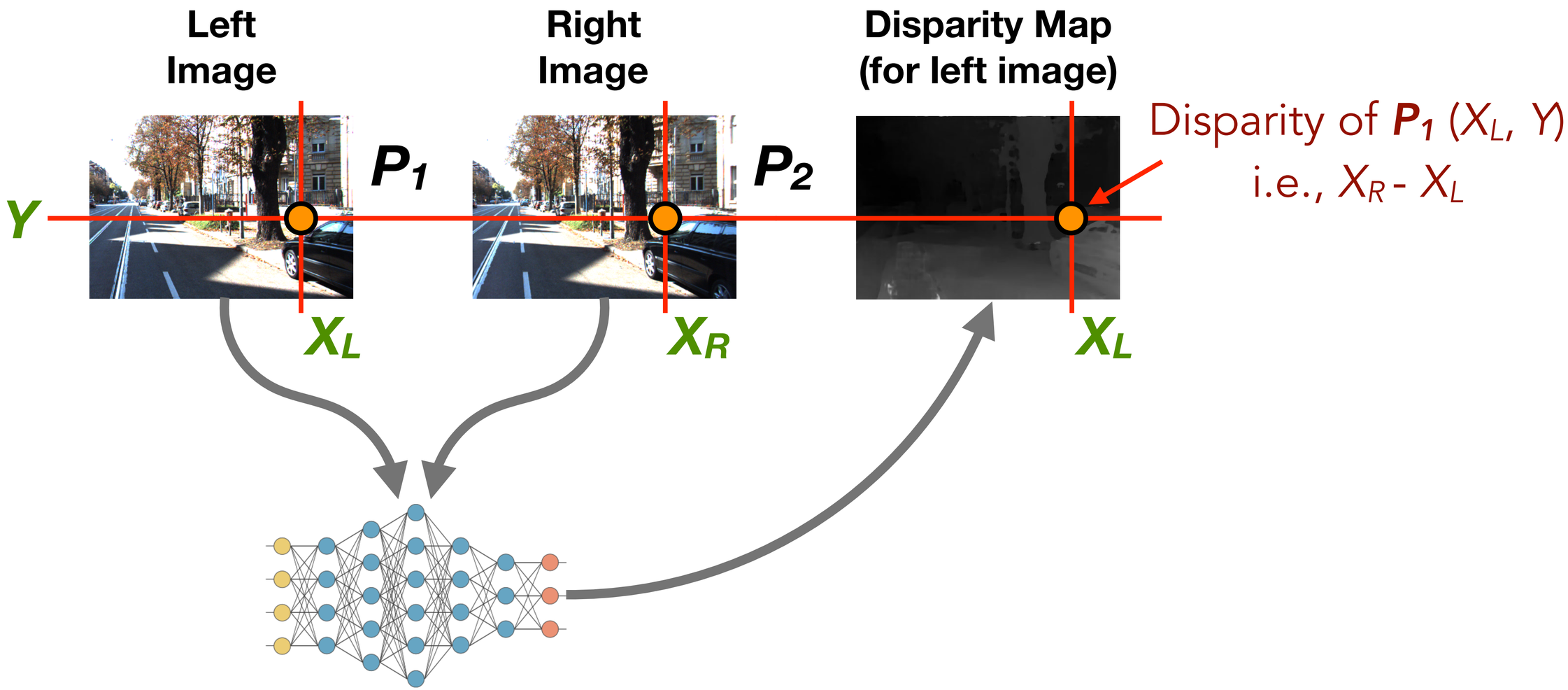


Disparity of P_1 (X_L, Y)
i.e., $X_R - X_L$

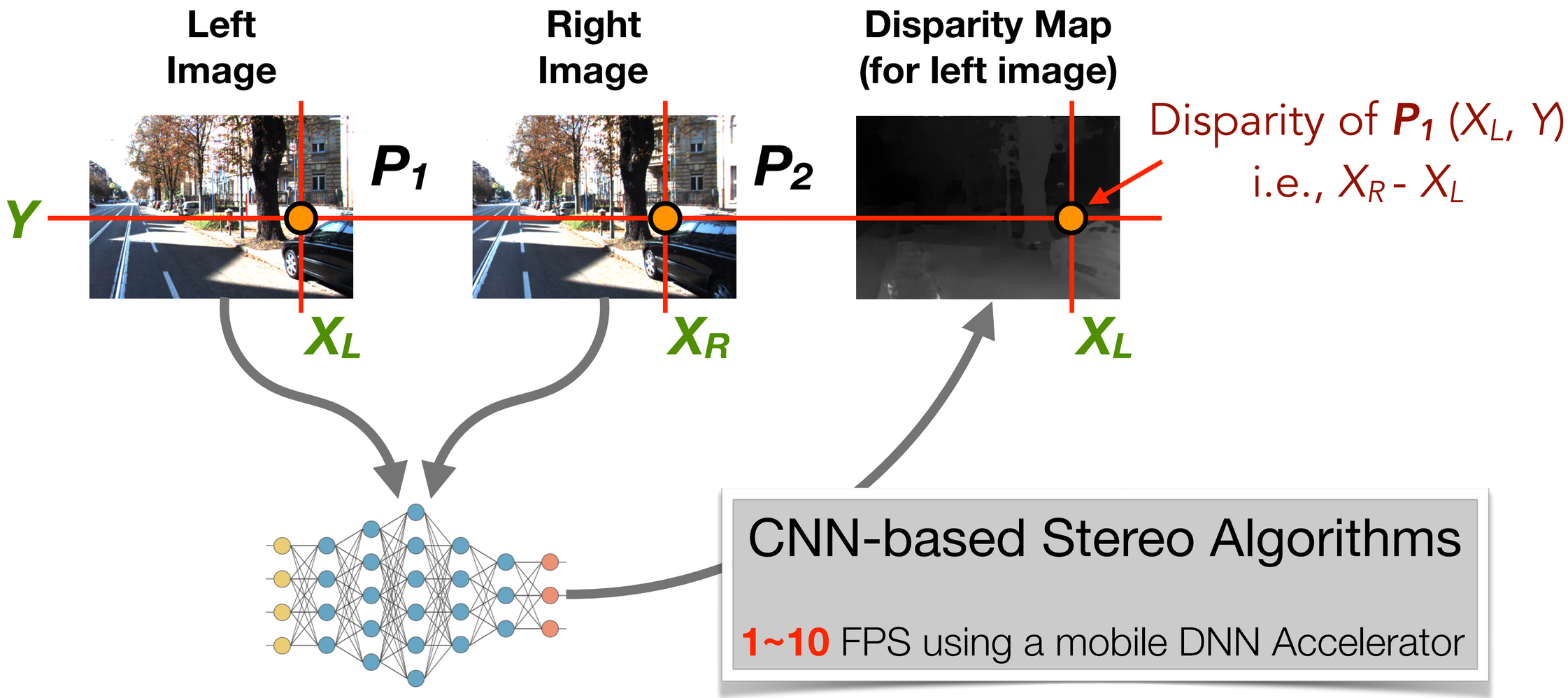
A Binocular Example: Depth Sensing



A Binocular Example: Depth Sensing



A Binocular Example: Depth Sensing



Depth Sensing Using Incremental Computing

	Left Image	Right Image	Disparity Map (for left image)
t_0			

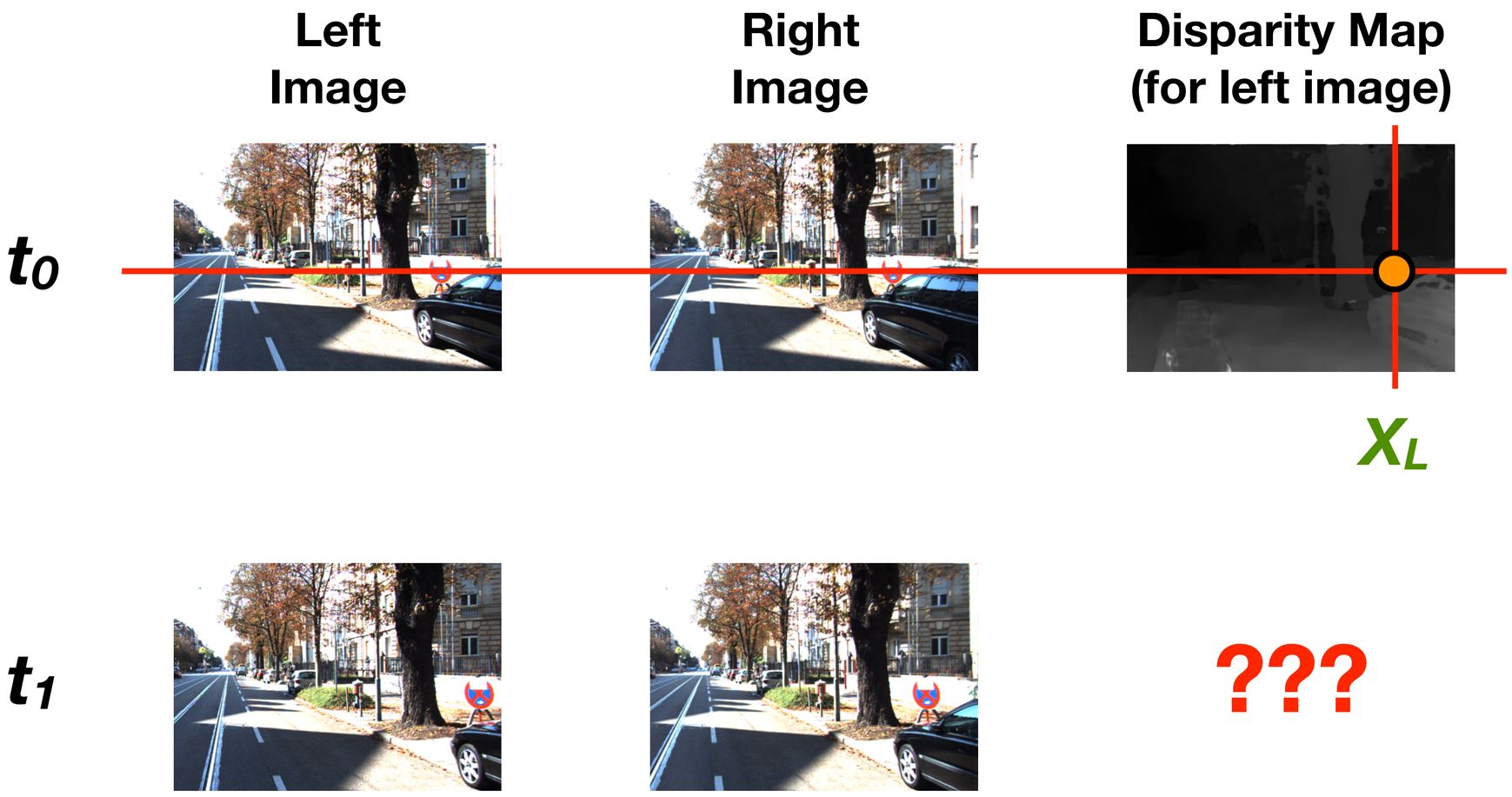
Depth Sensing Using Incremental Computing

	Left Image	Right Image	Disparity Map (for left image)
t_0			
t_1			

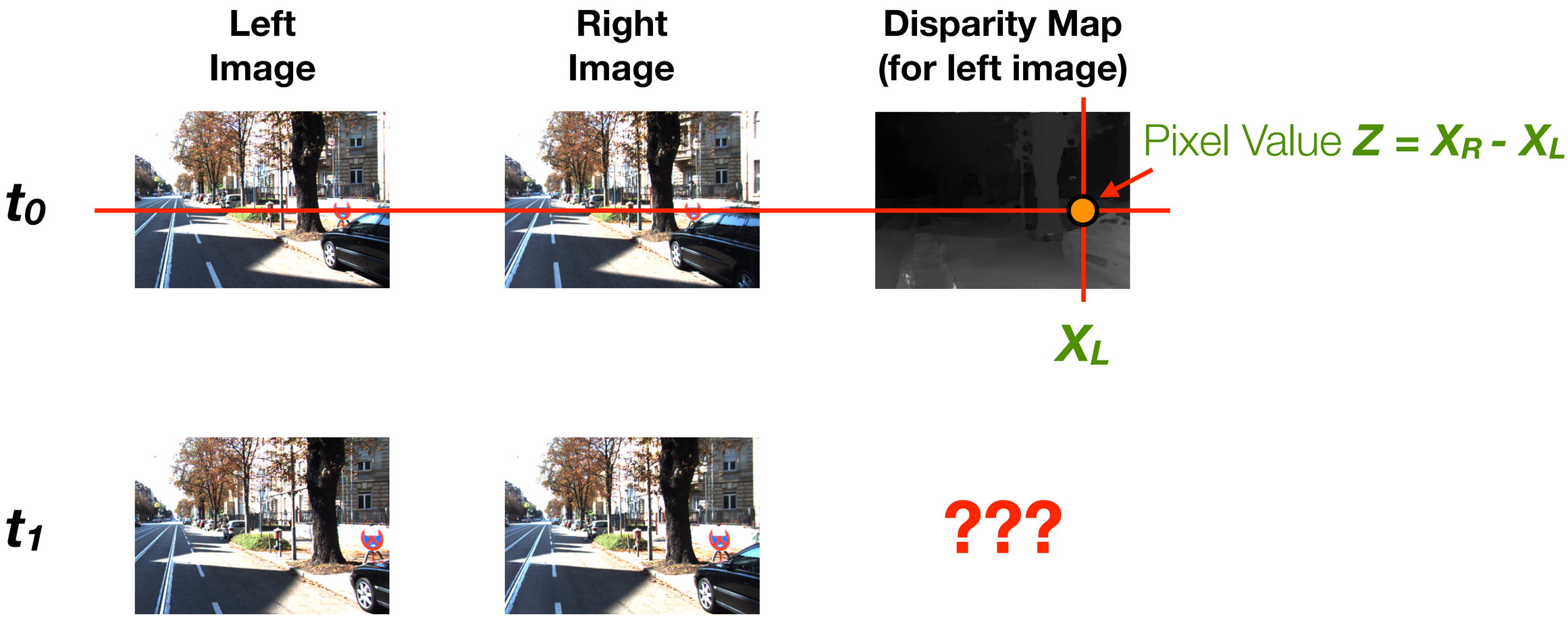
Depth Sensing Using Incremental Computing

	Left Image	Right Image	Disparity Map (for left image)
t_0			
t_1			???

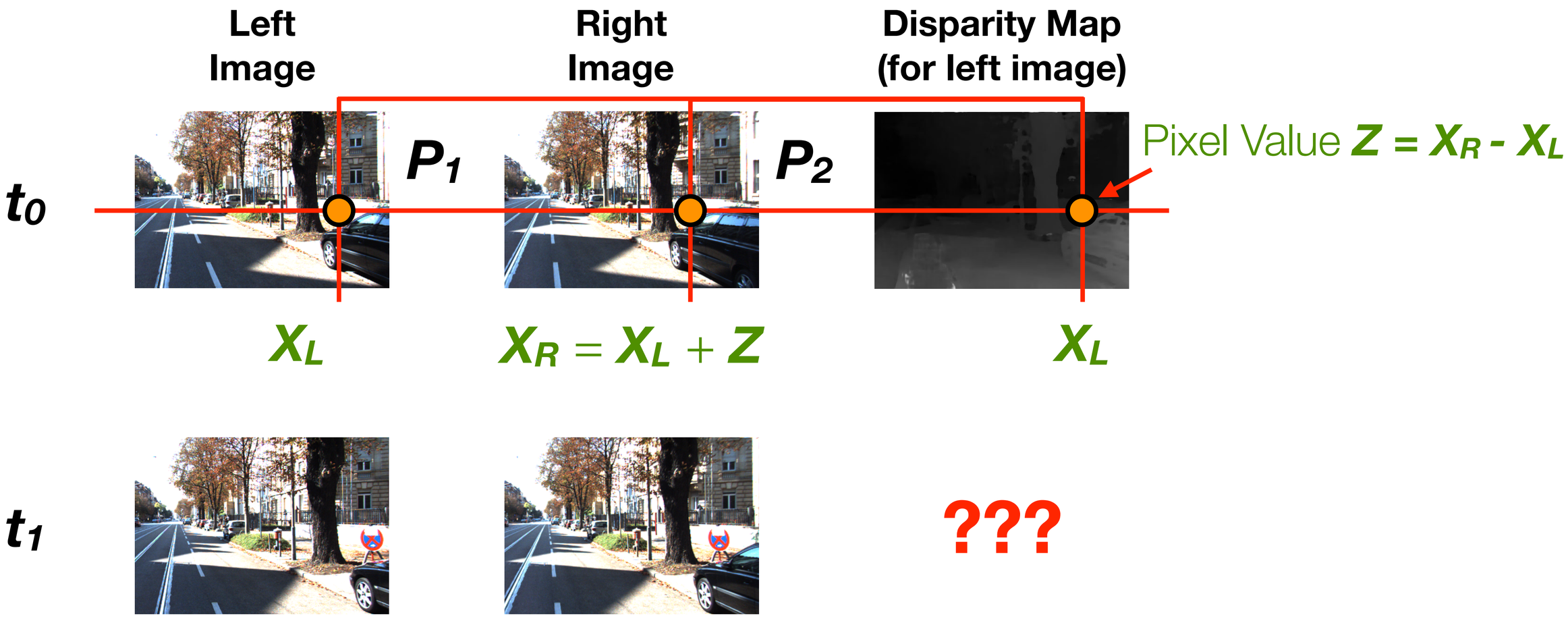
Depth Sensing Using Incremental Computing



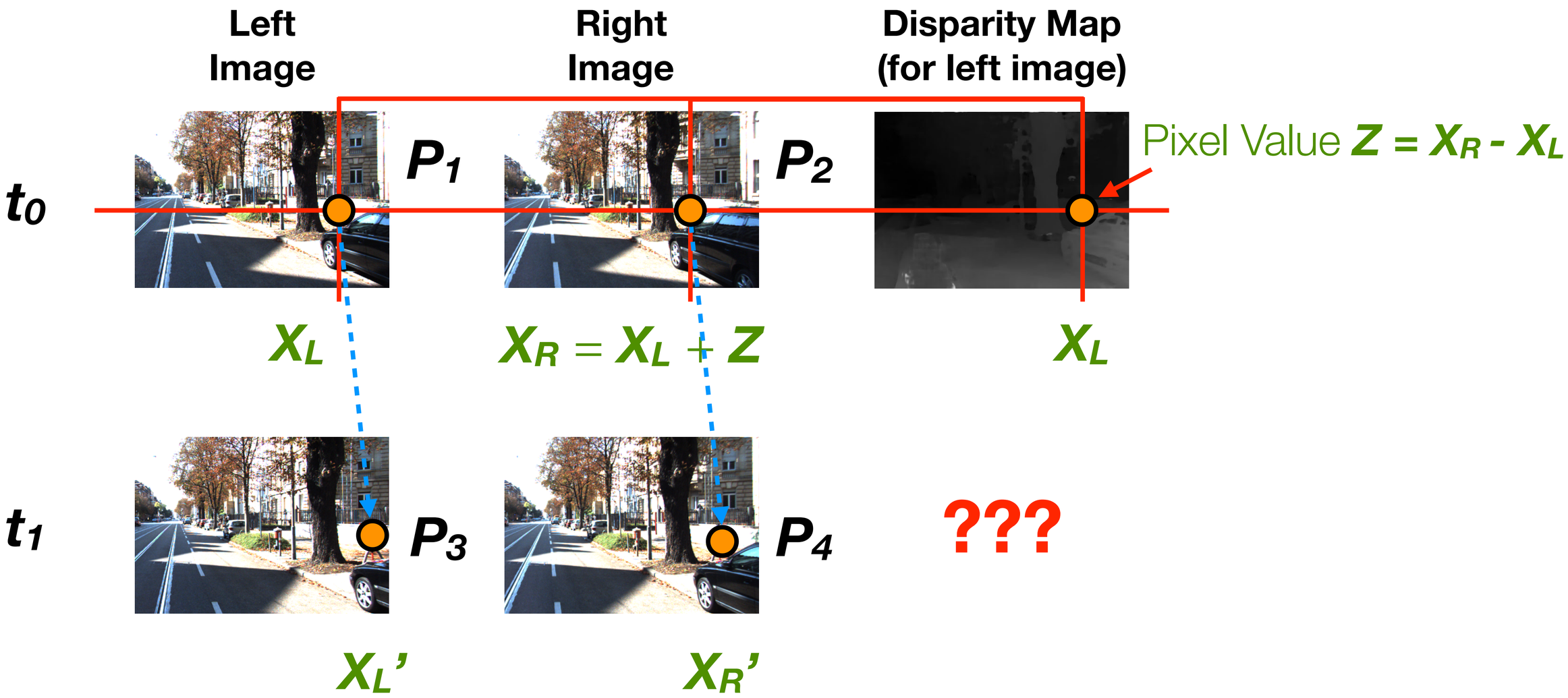
Depth Sensing Using Incremental Computing



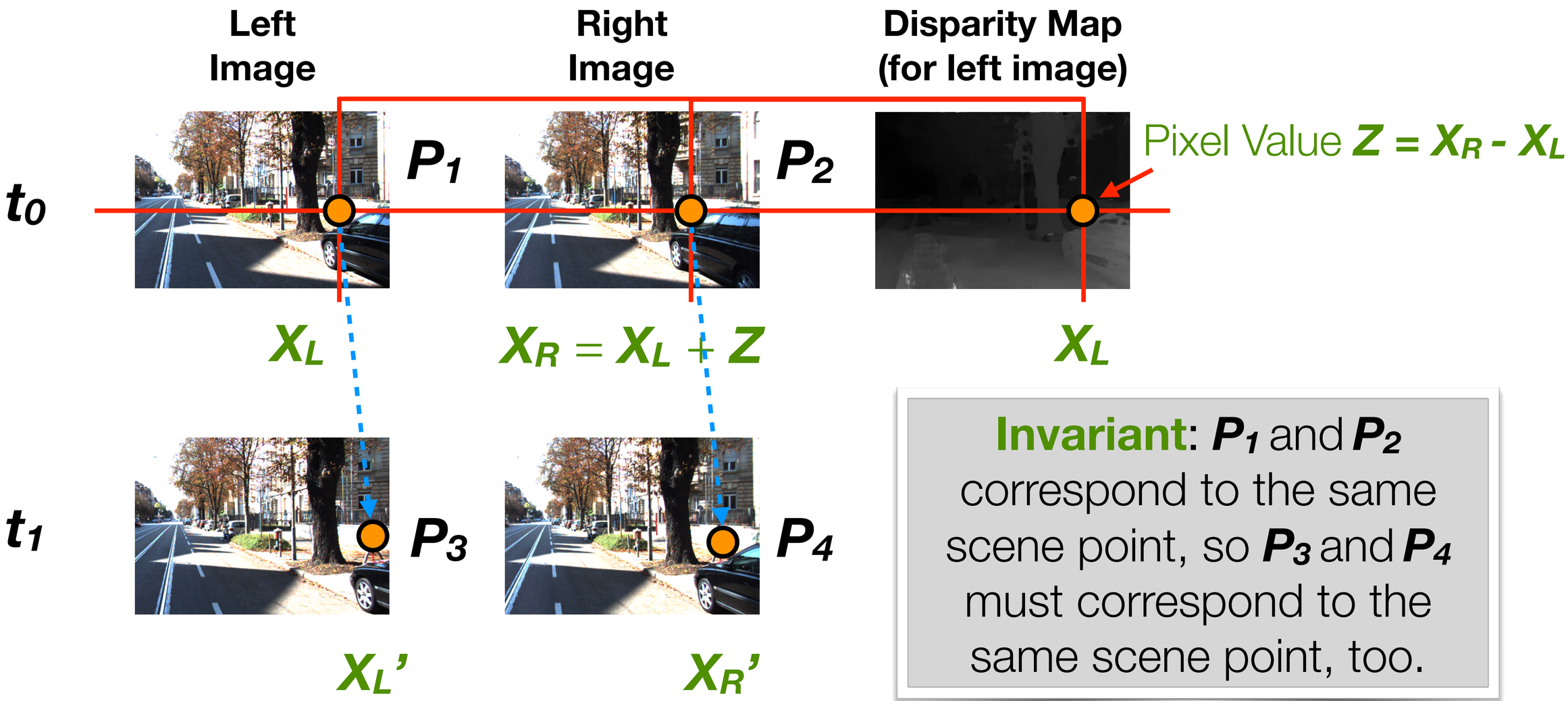
Depth Sensing Using Incremental Computing



Depth Sensing Using Incremental Computing

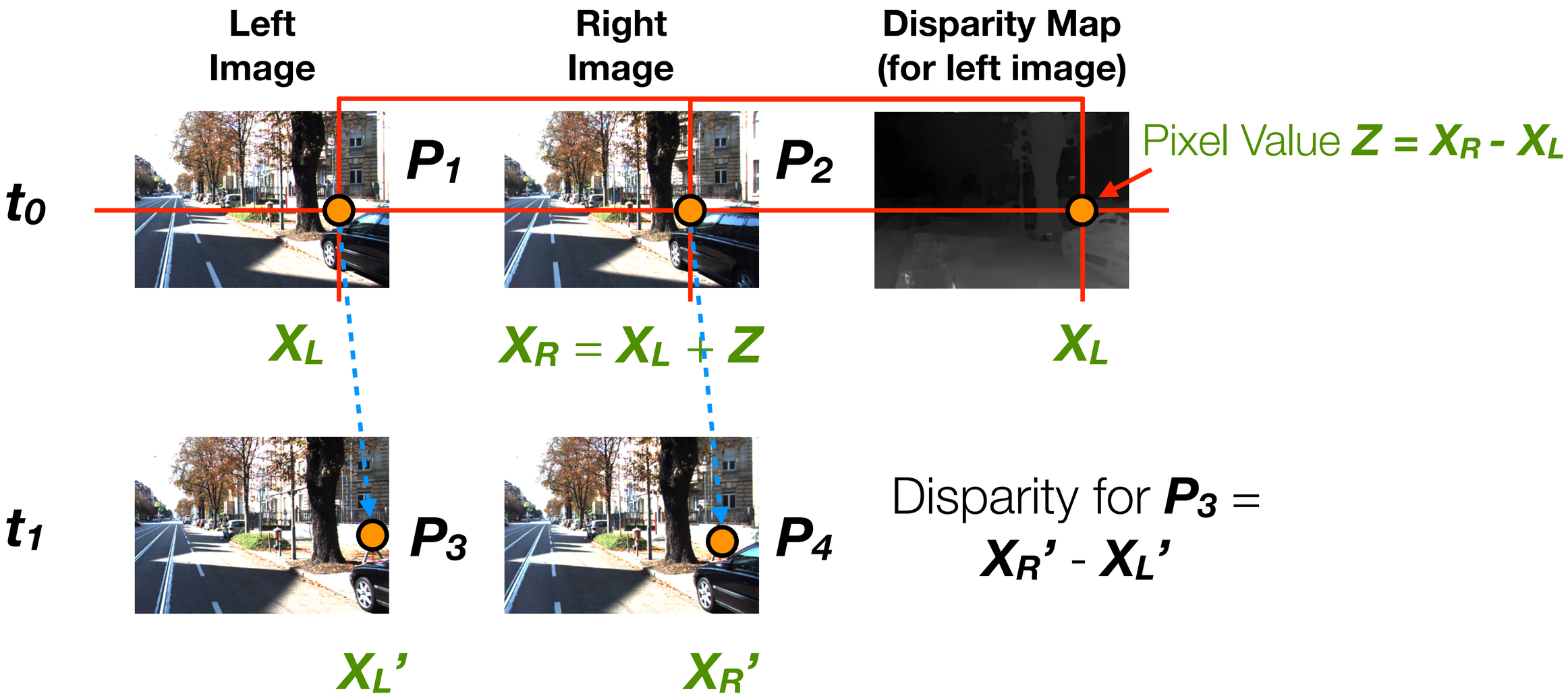


Depth Sensing Using Incremental Computing



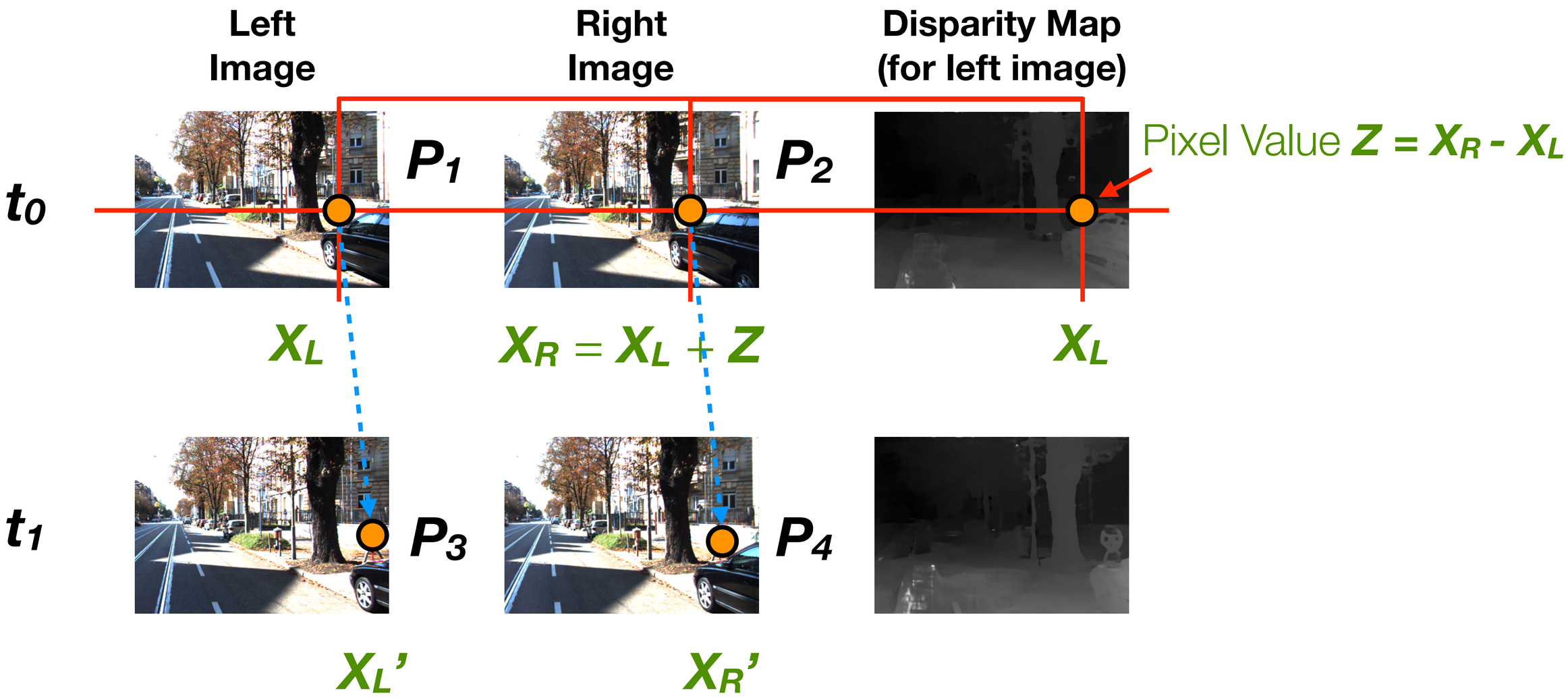
Invariant: P_1 and P_2 correspond to the same scene point, so P_3 and P_4 must correspond to the same scene point, too.

Depth Sensing Using Incremental Computing

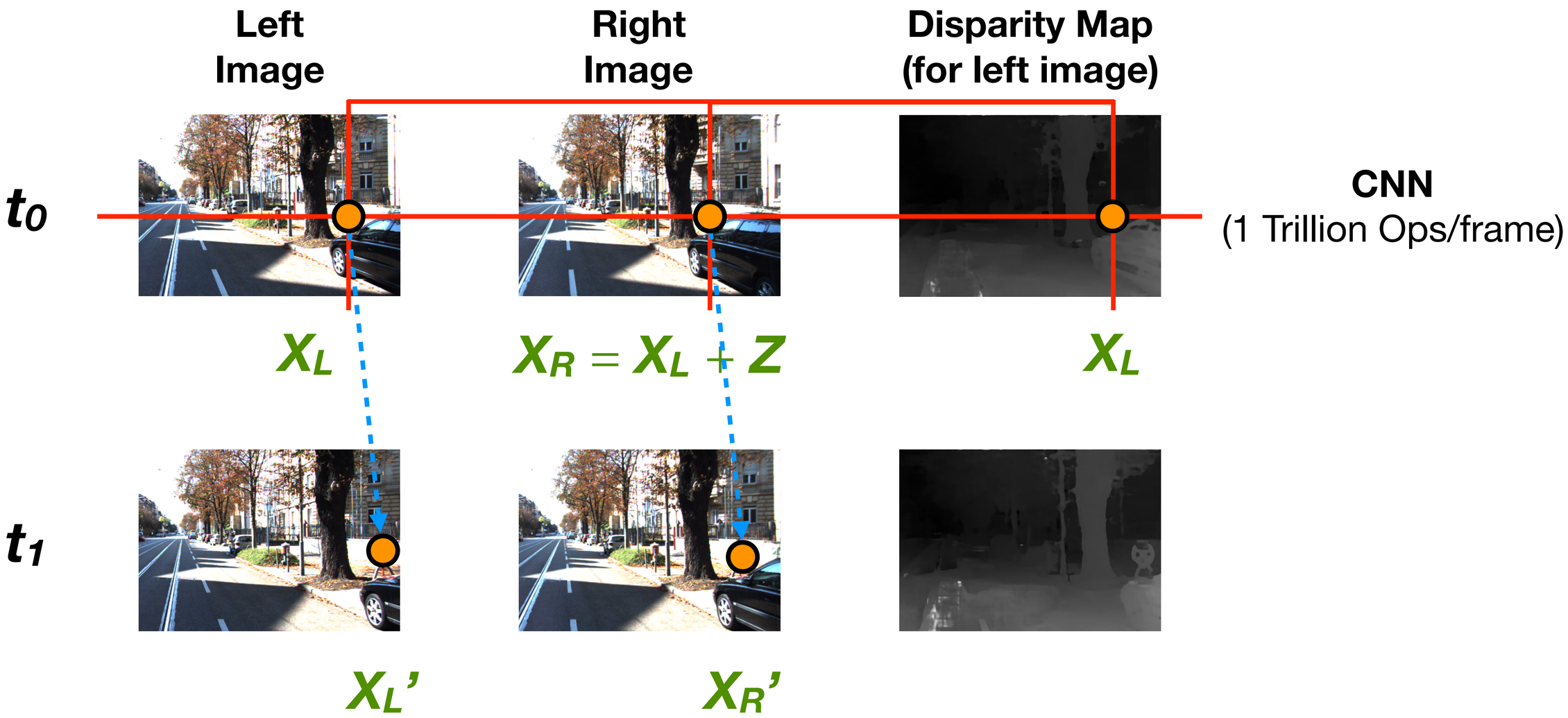


Disparity for P_3 = $X_R' - X_L'$

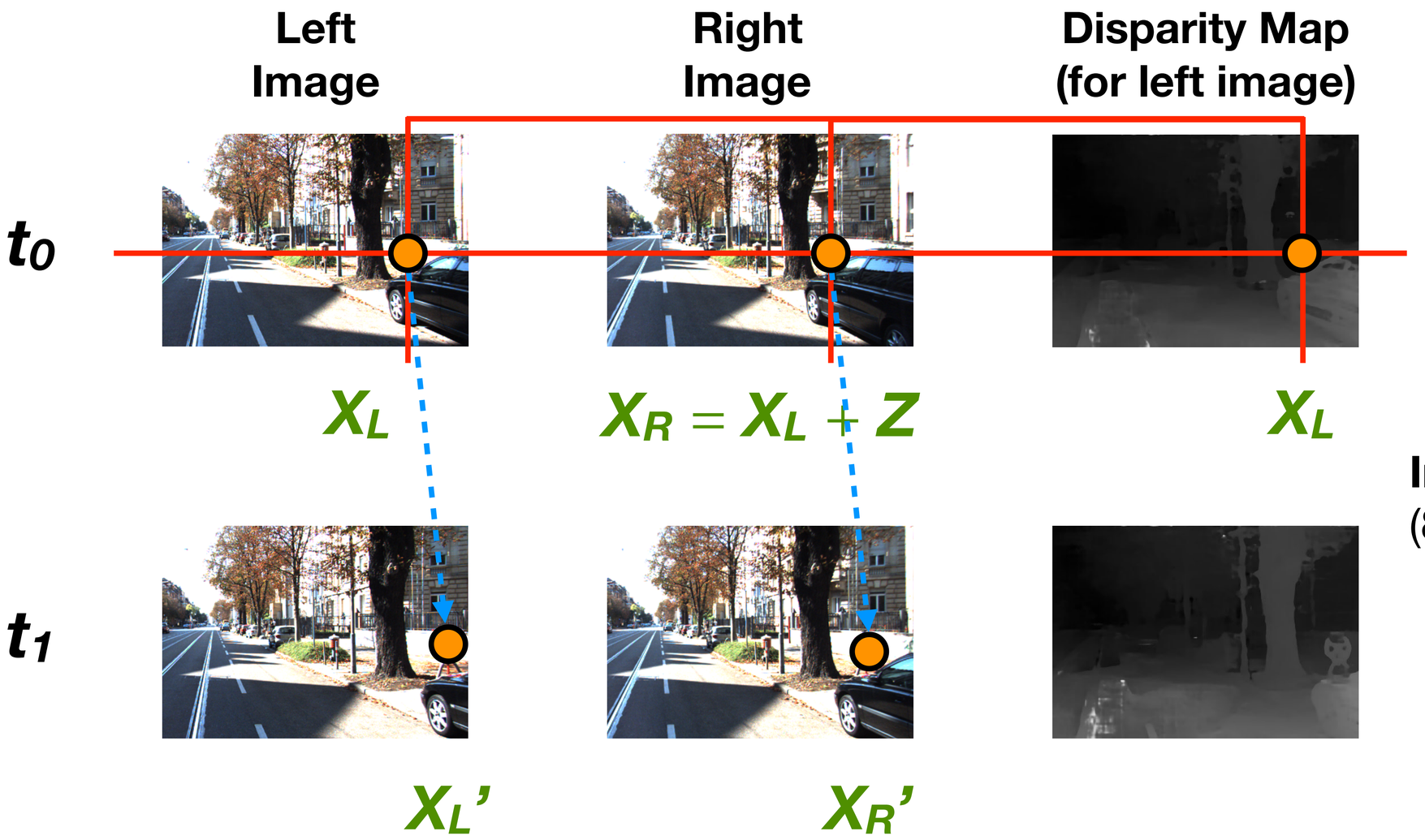
Depth Sensing Using Incremental Computing



Depth Sensing Using Incremental Computing



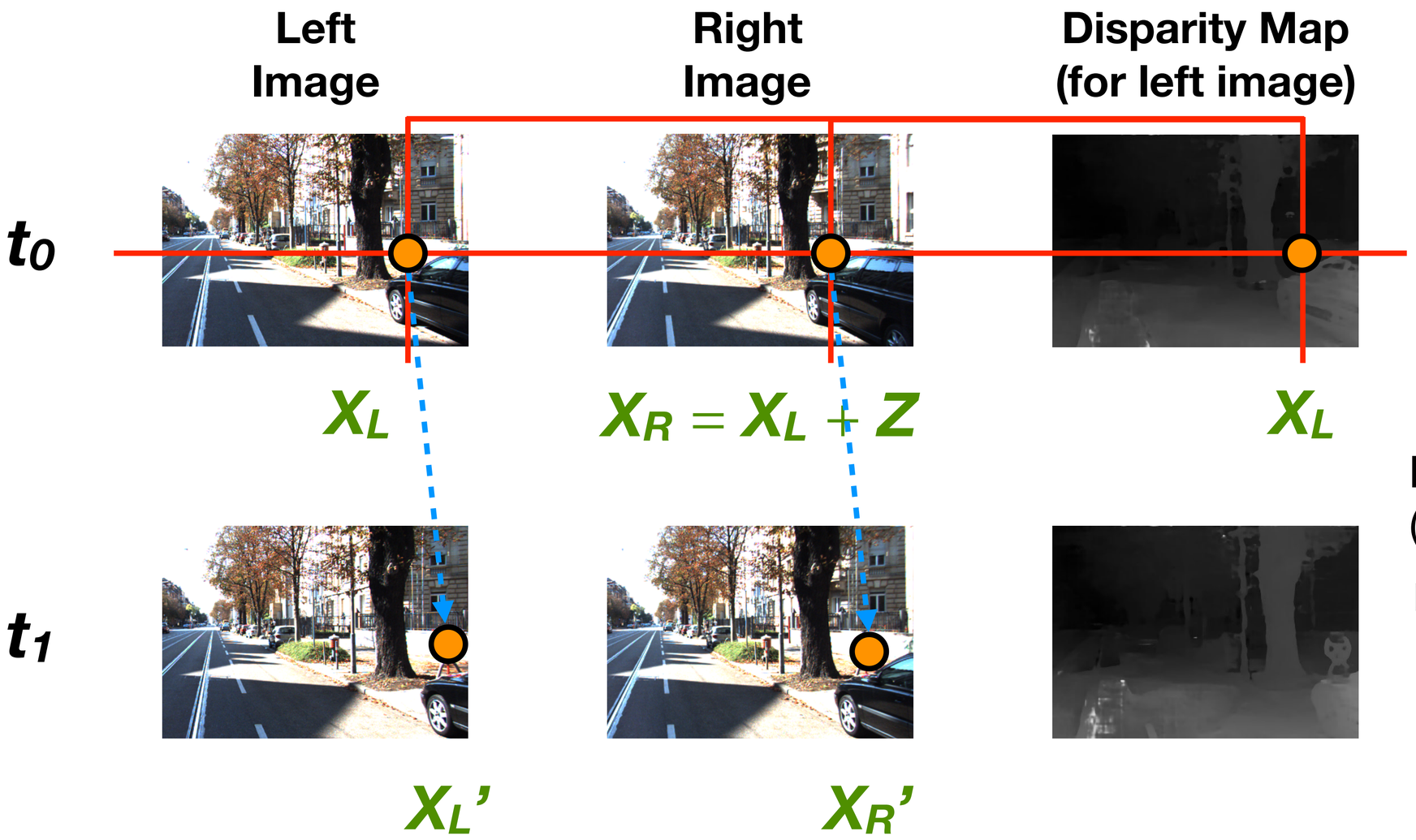
Depth Sensing Using Incremental Computing



CNN
(1 Trillion Ops/frame)

Incremental Computing
(87 Million Ops/Frame)

Depth Sensing Using Incremental Computing

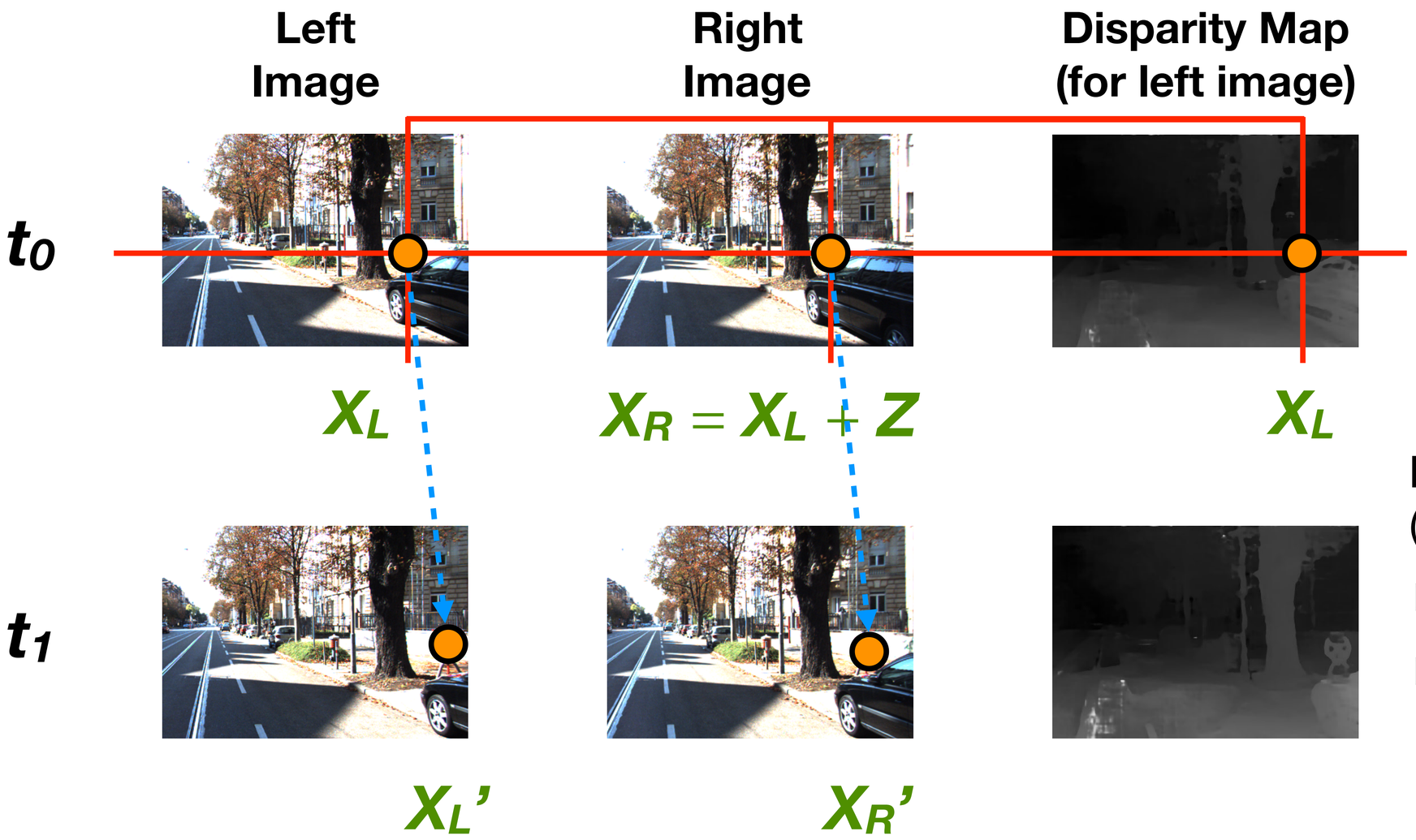


CNN
(1 Trillion Ops/frame)

Incremental Computing
(87 Million Ops/Frame)

- ⊖ Motion vector

Depth Sensing Using Incremental Computing

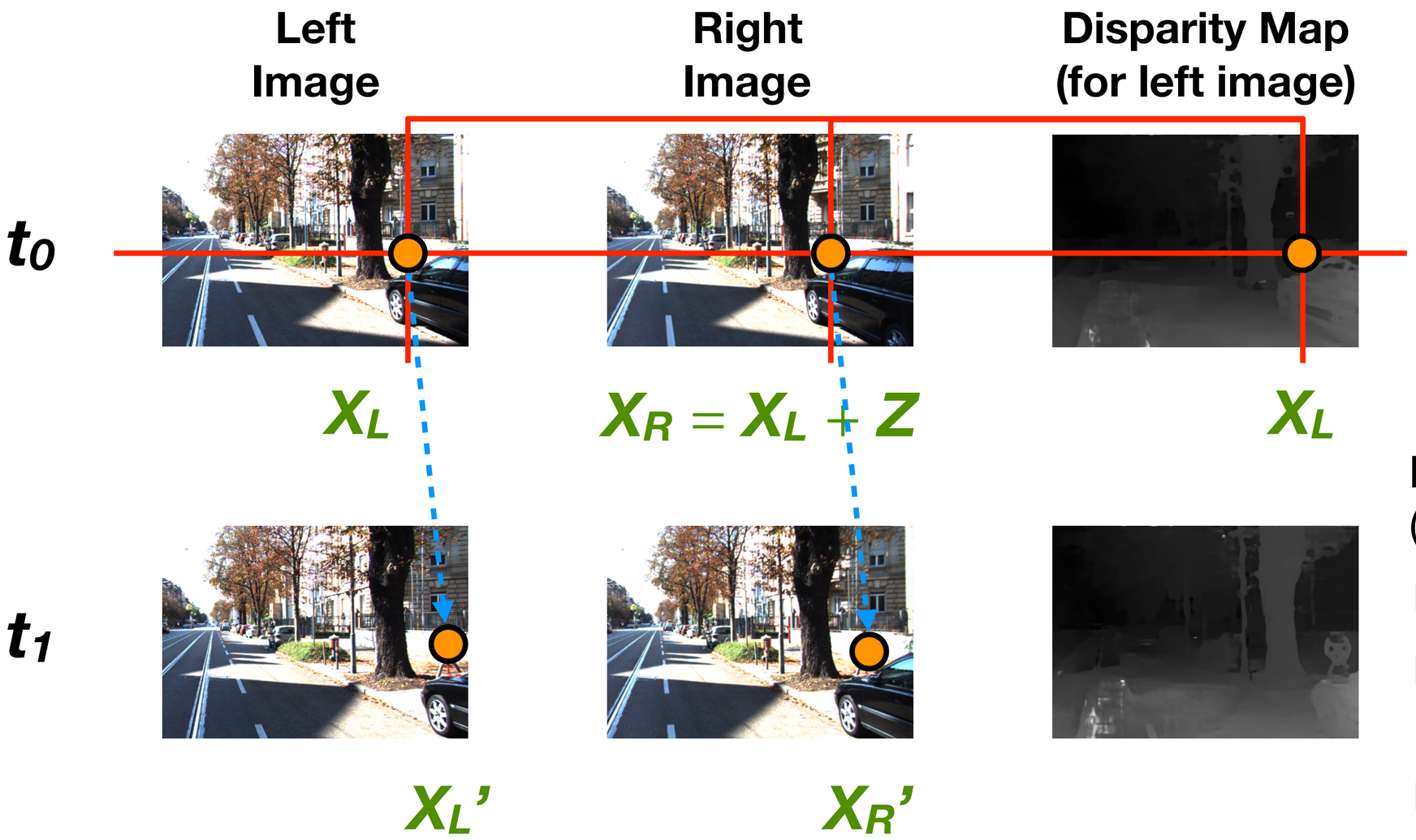


CNN
(1 Trillion Ops/frame)

Incremental Computing
(87 Million Ops/Frame)

- ⊖ Motion vector
- ***df*** Reconstruct corresponding pixels

Depth Sensing Using Incremental Computing

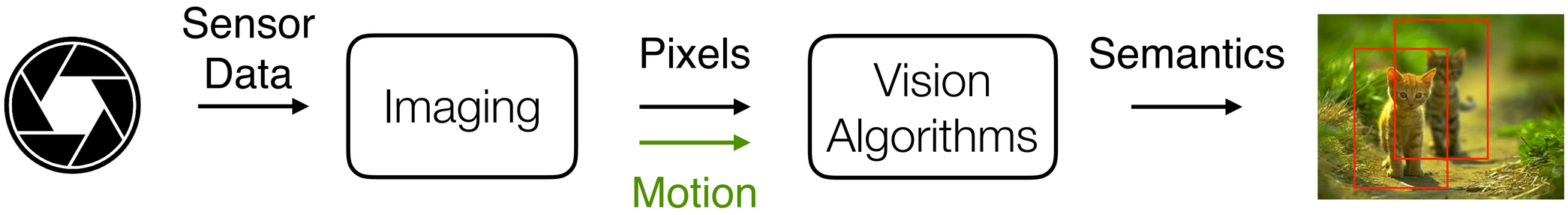


CNN
(1 Trillion Ops/frame)

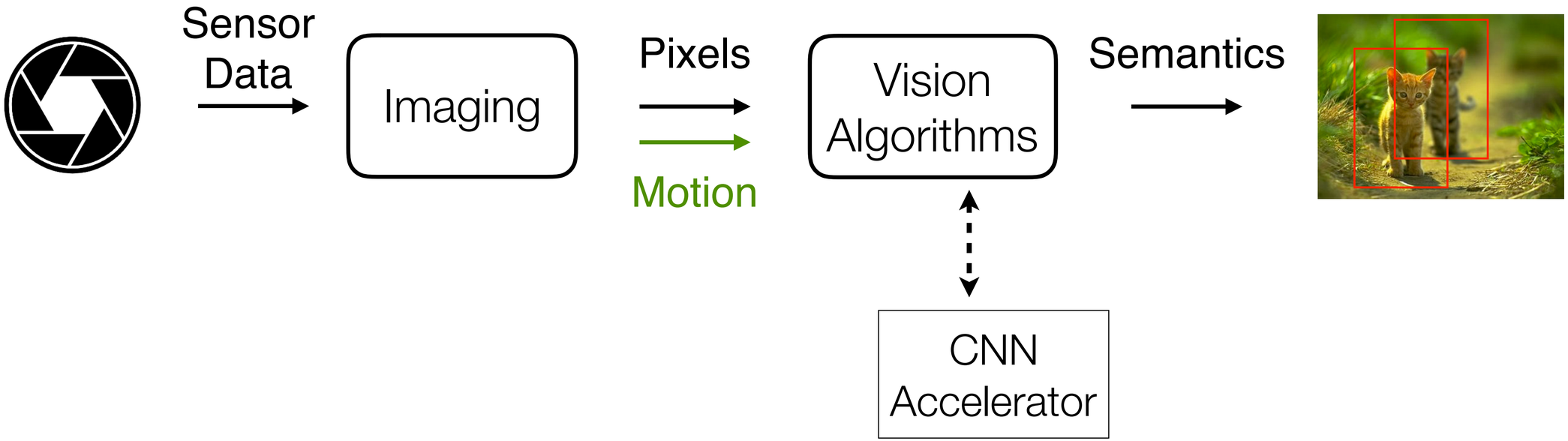
Incremental Computing
(87 Million Ops/Frame)

- ⊖ Motion vector
- ***df*** Reconstruct corresponding pixels
- ⊕ Triangulation

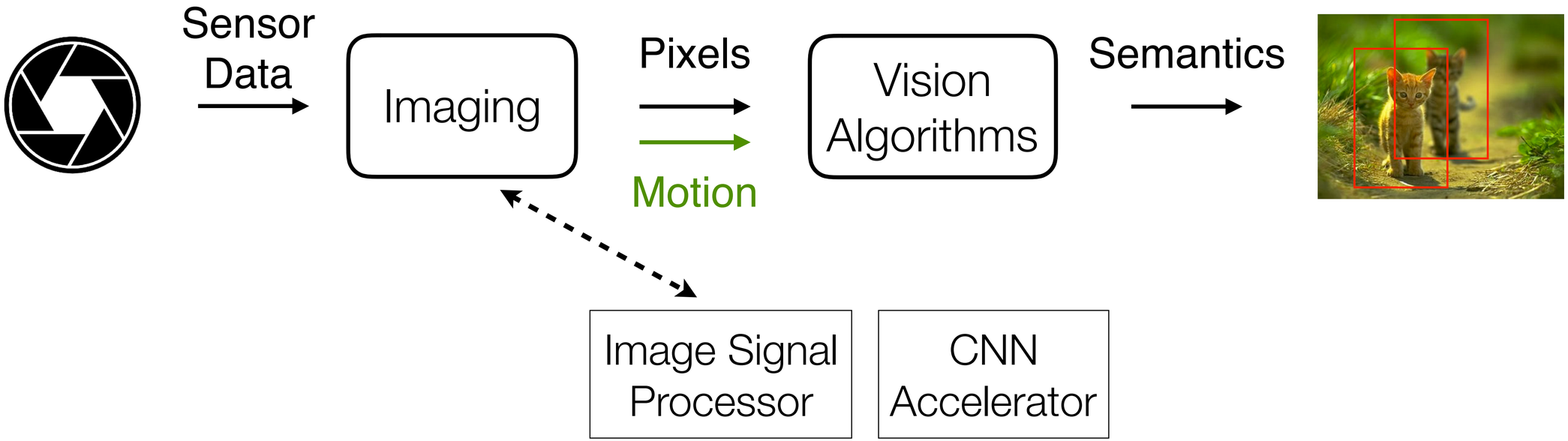
Hardware Architecture Support



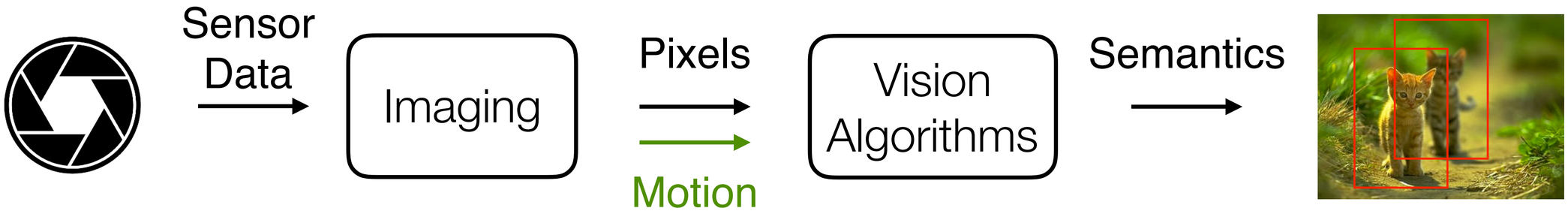
Hardware Architecture Support

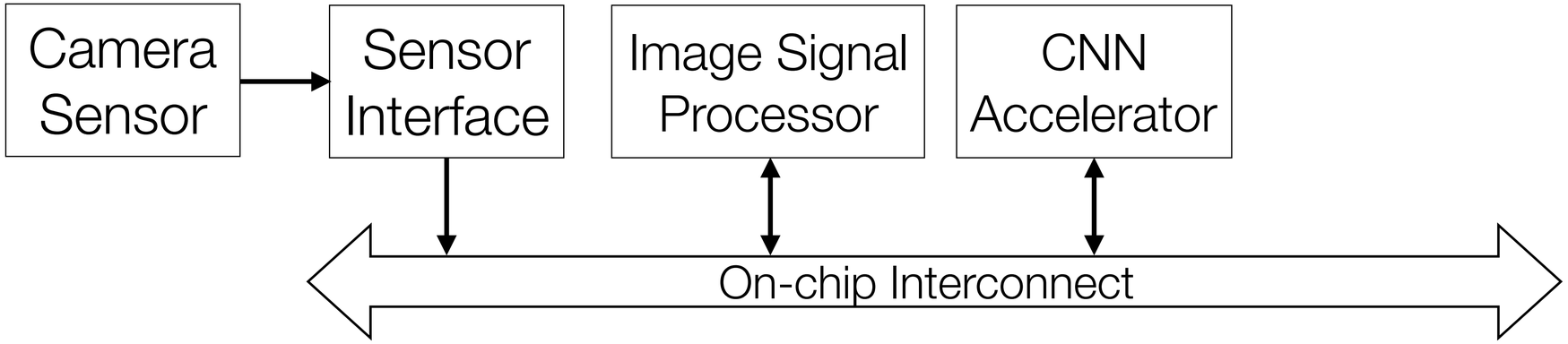


Hardware Architecture Support

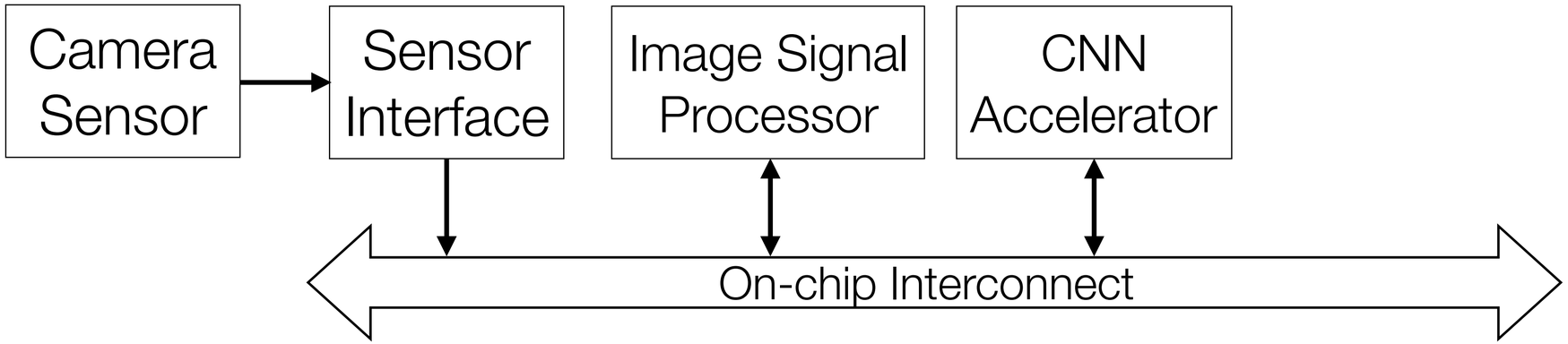


Hardware Architecture Support

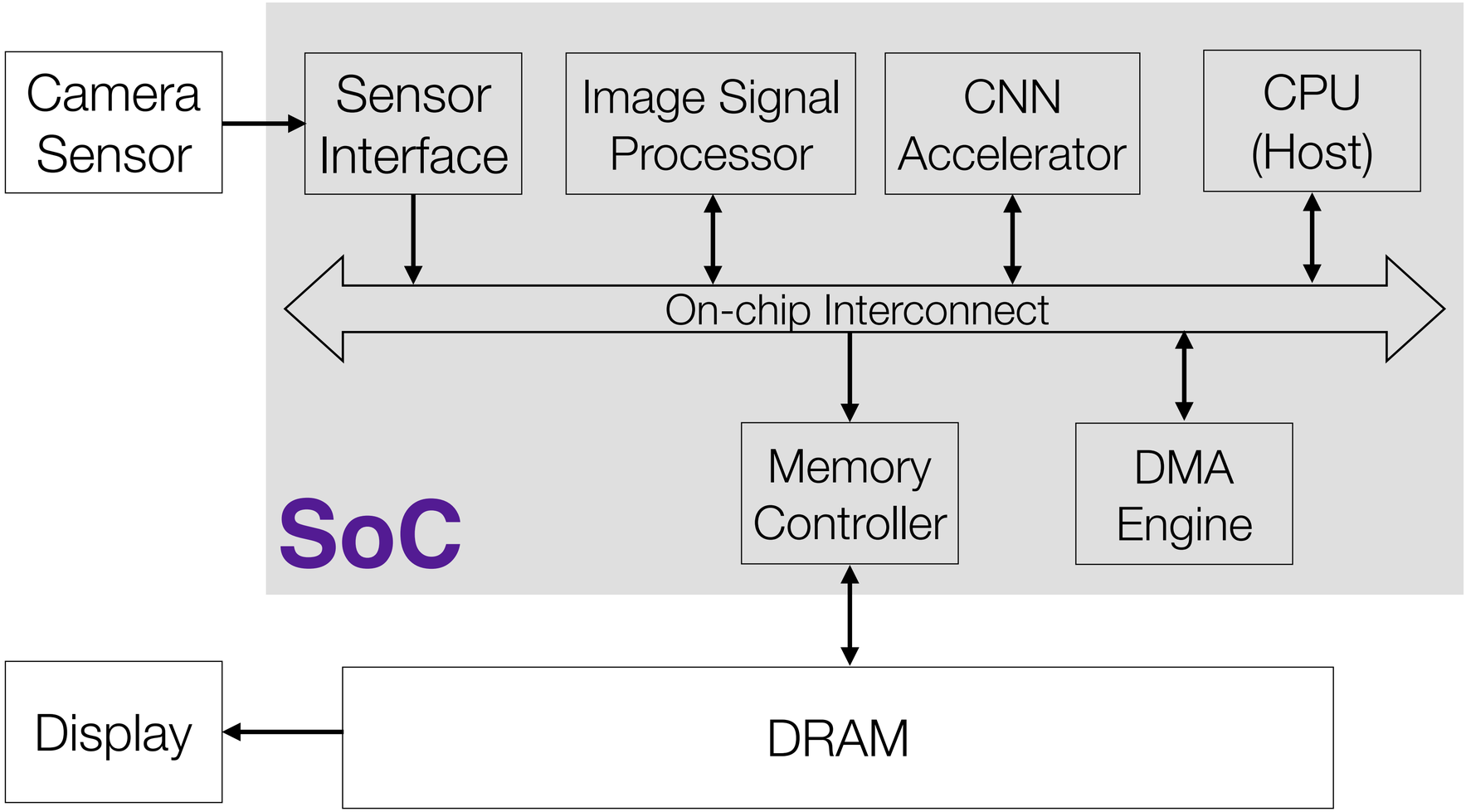




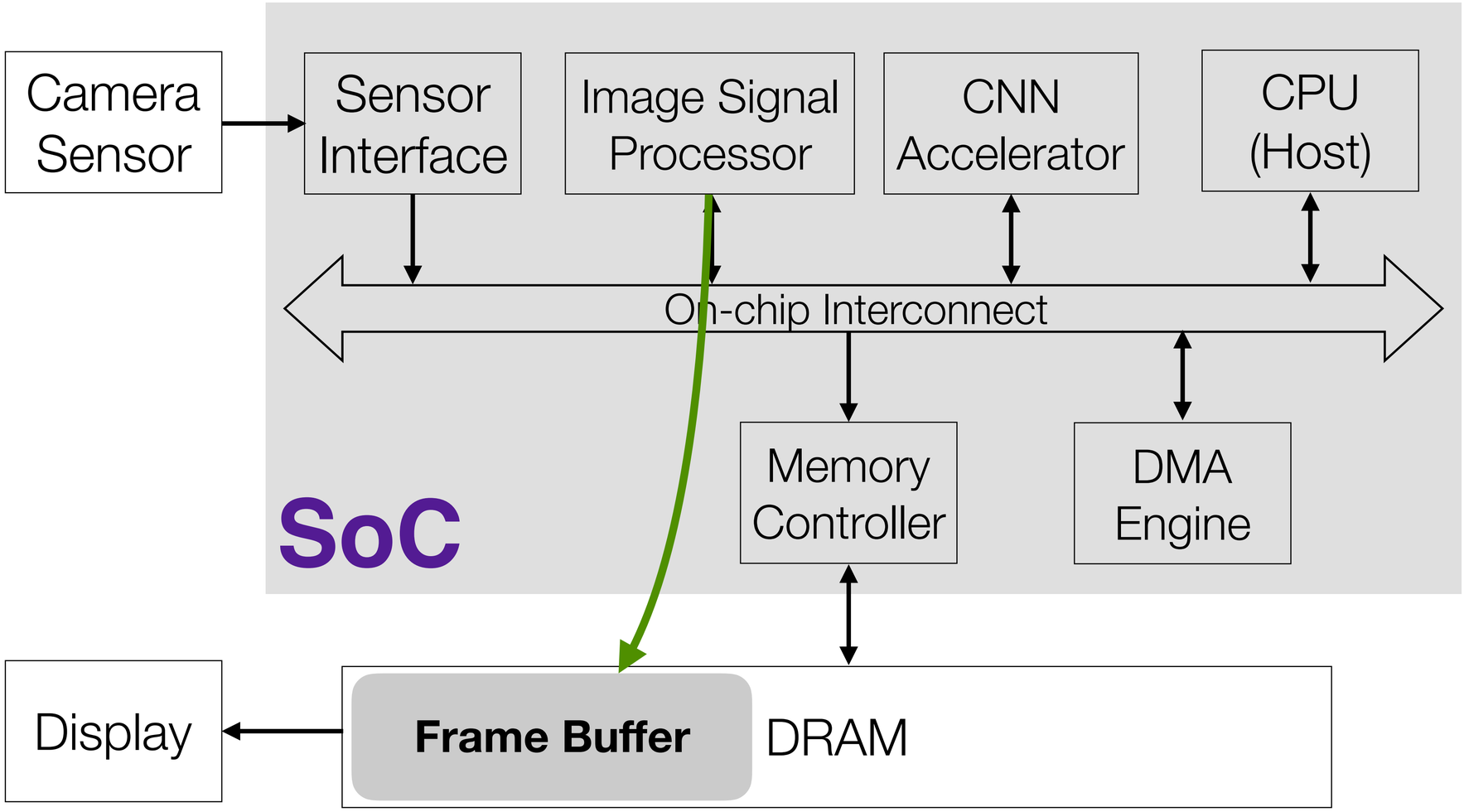
Hardware Architecture Support



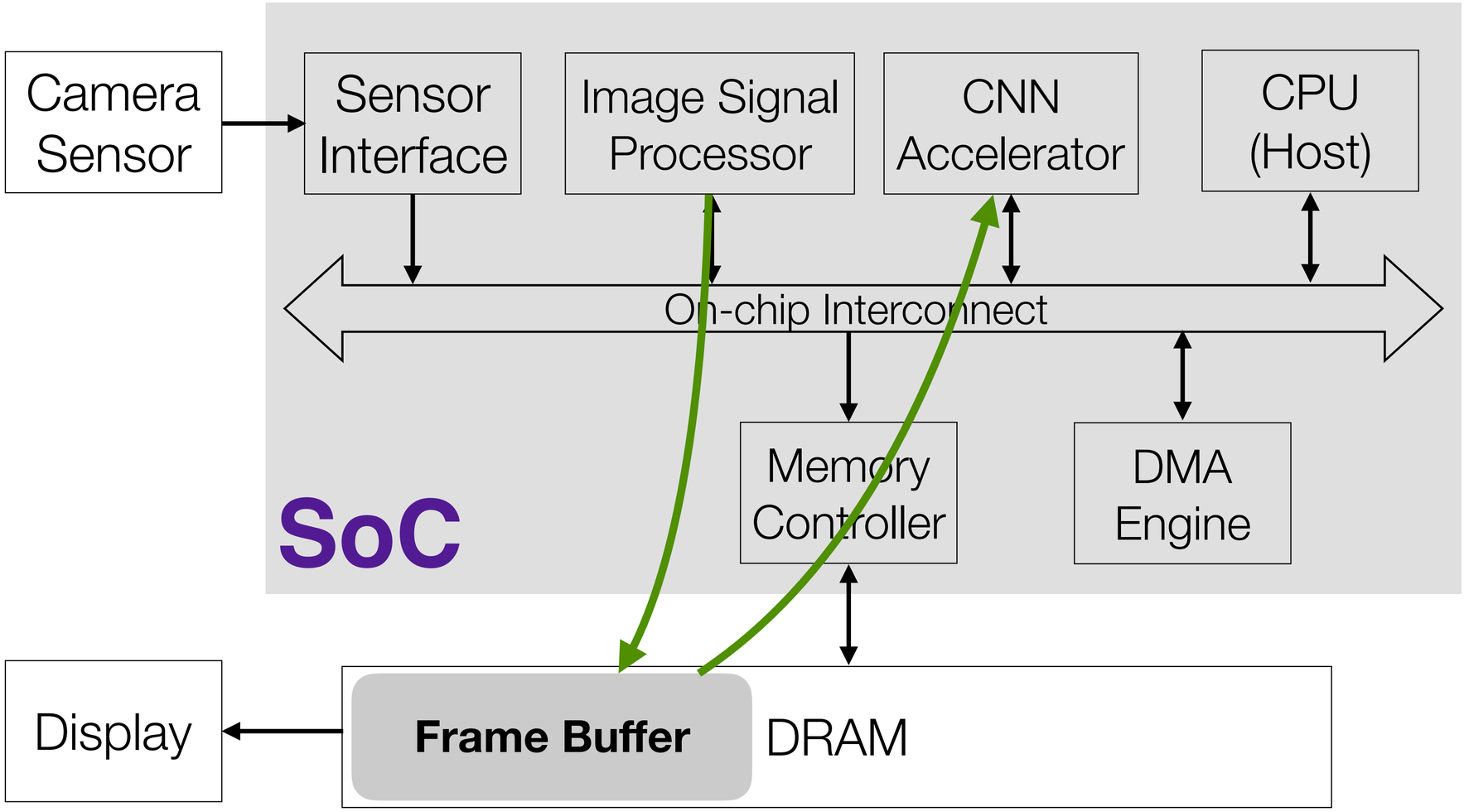
Hardware Architecture Support



Hardware Architecture Support

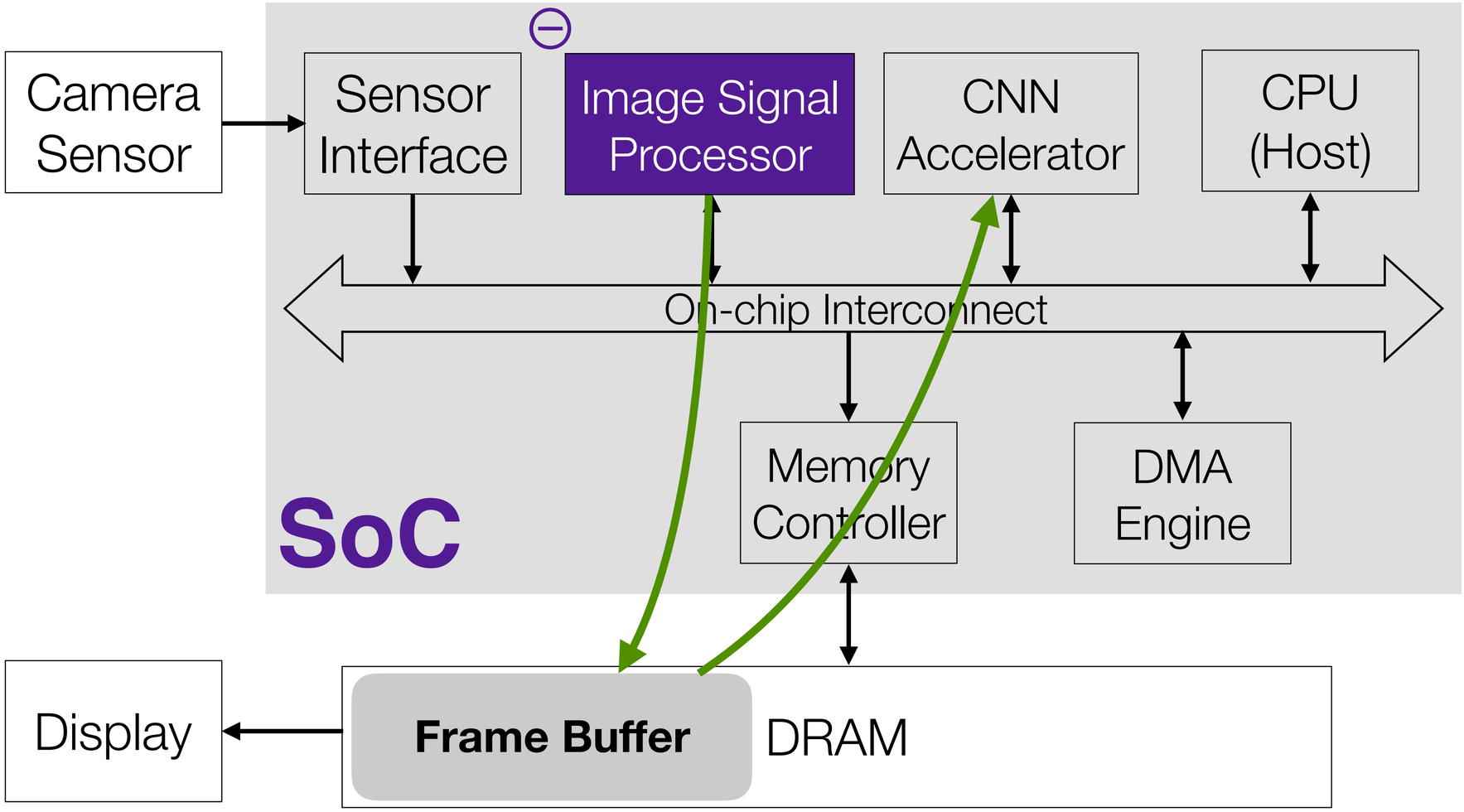


Hardware Architecture Support



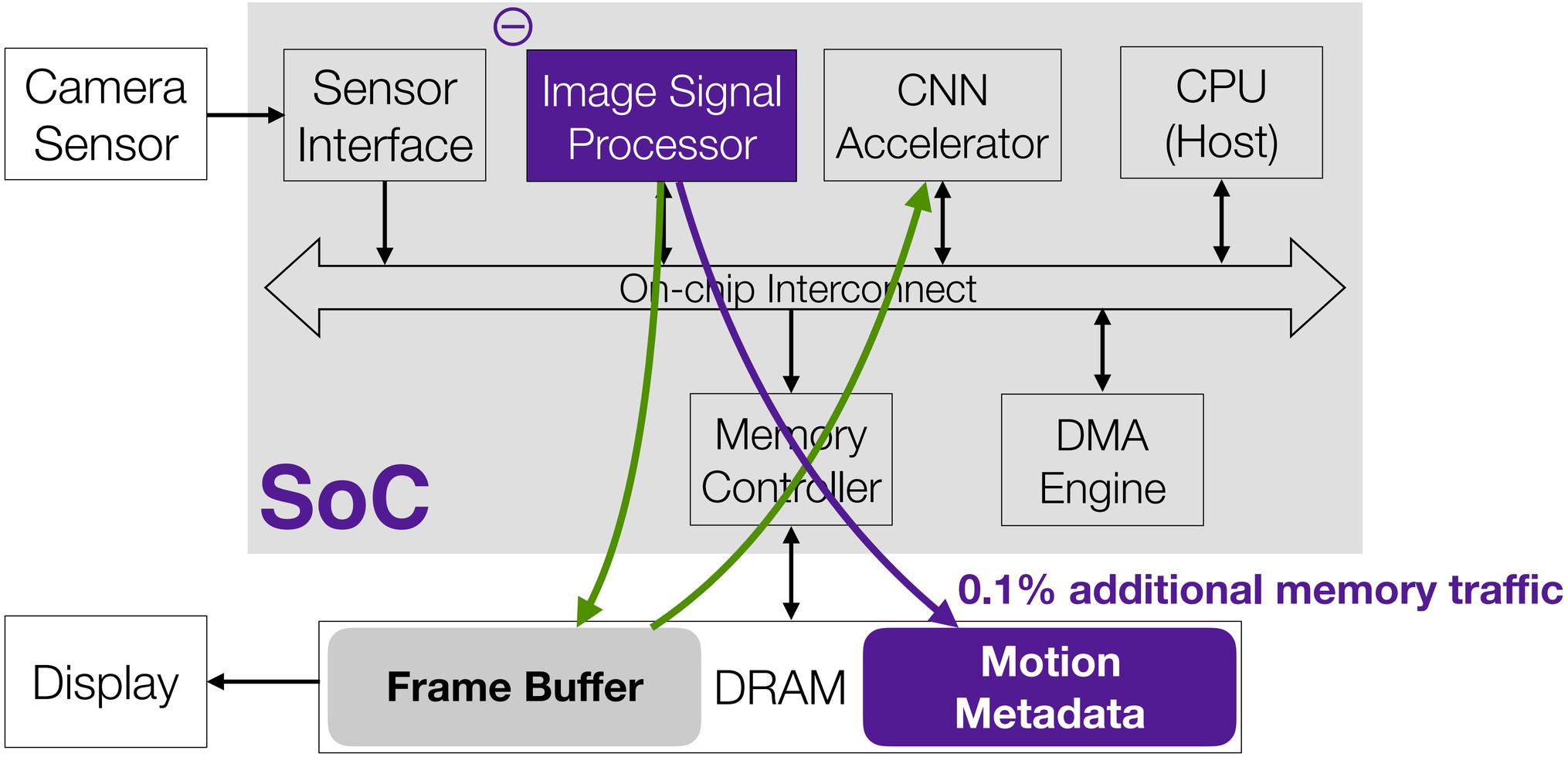
Hardware Architecture Support

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$



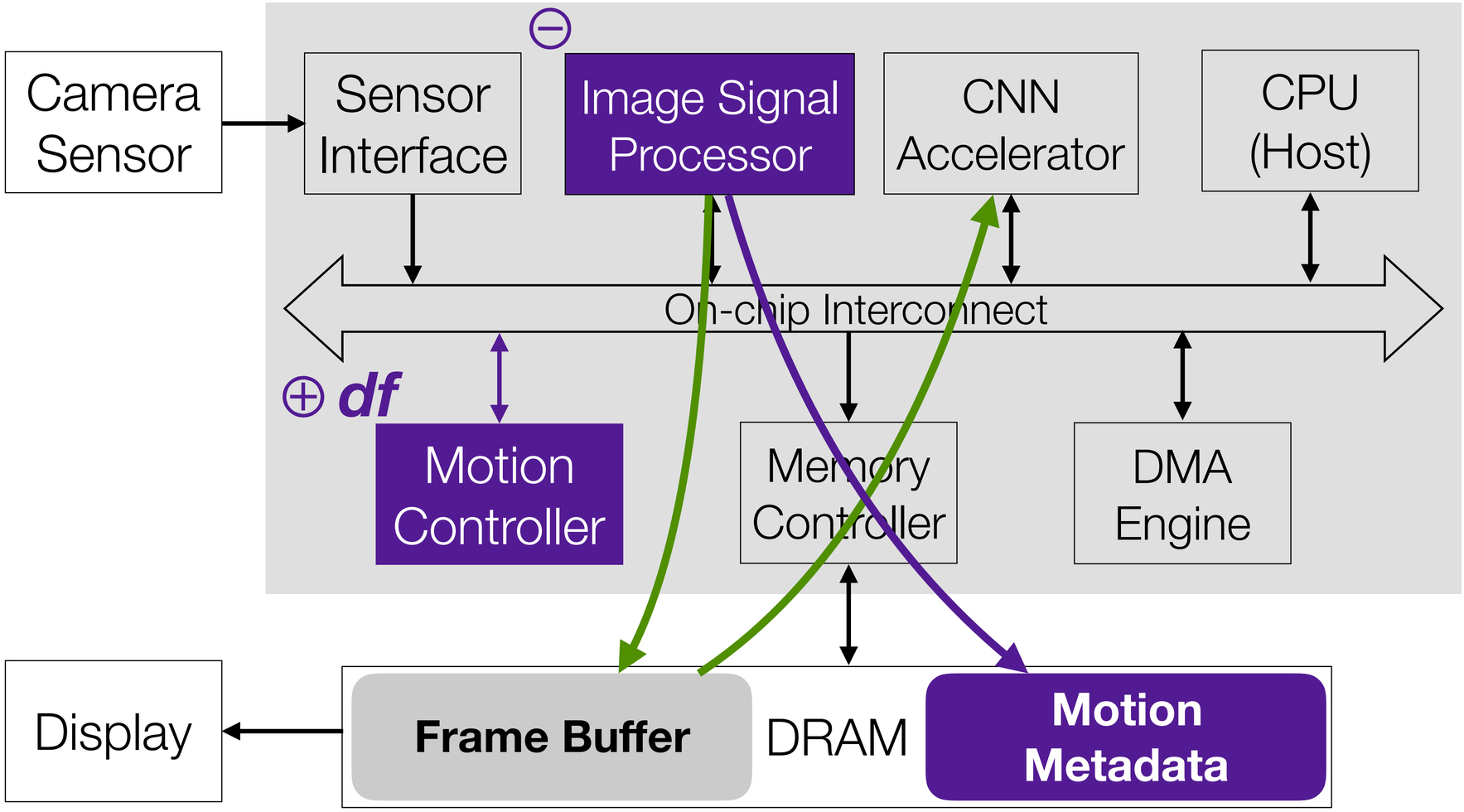
Hardware Architecture Support

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$



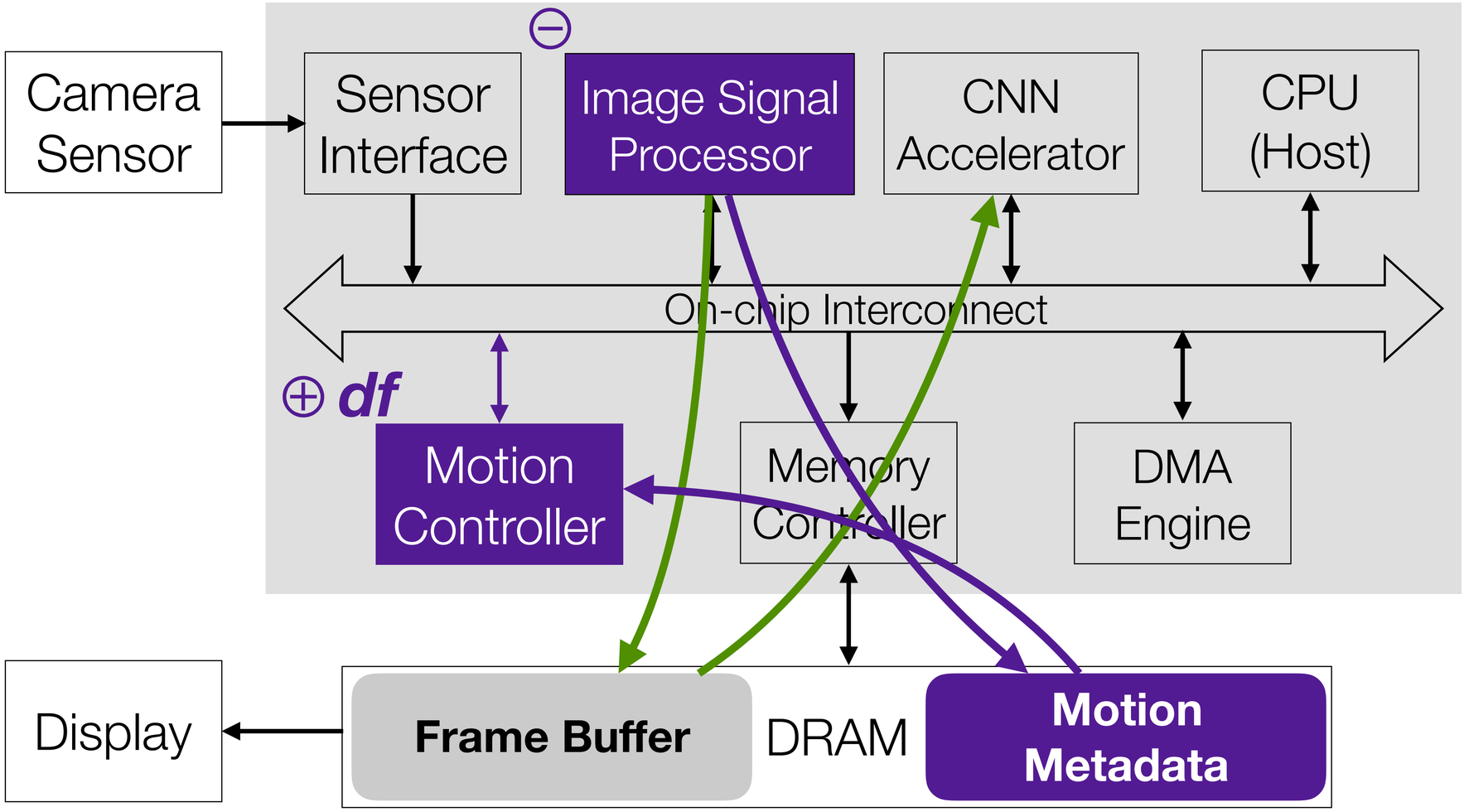
Hardware Architecture Support

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$



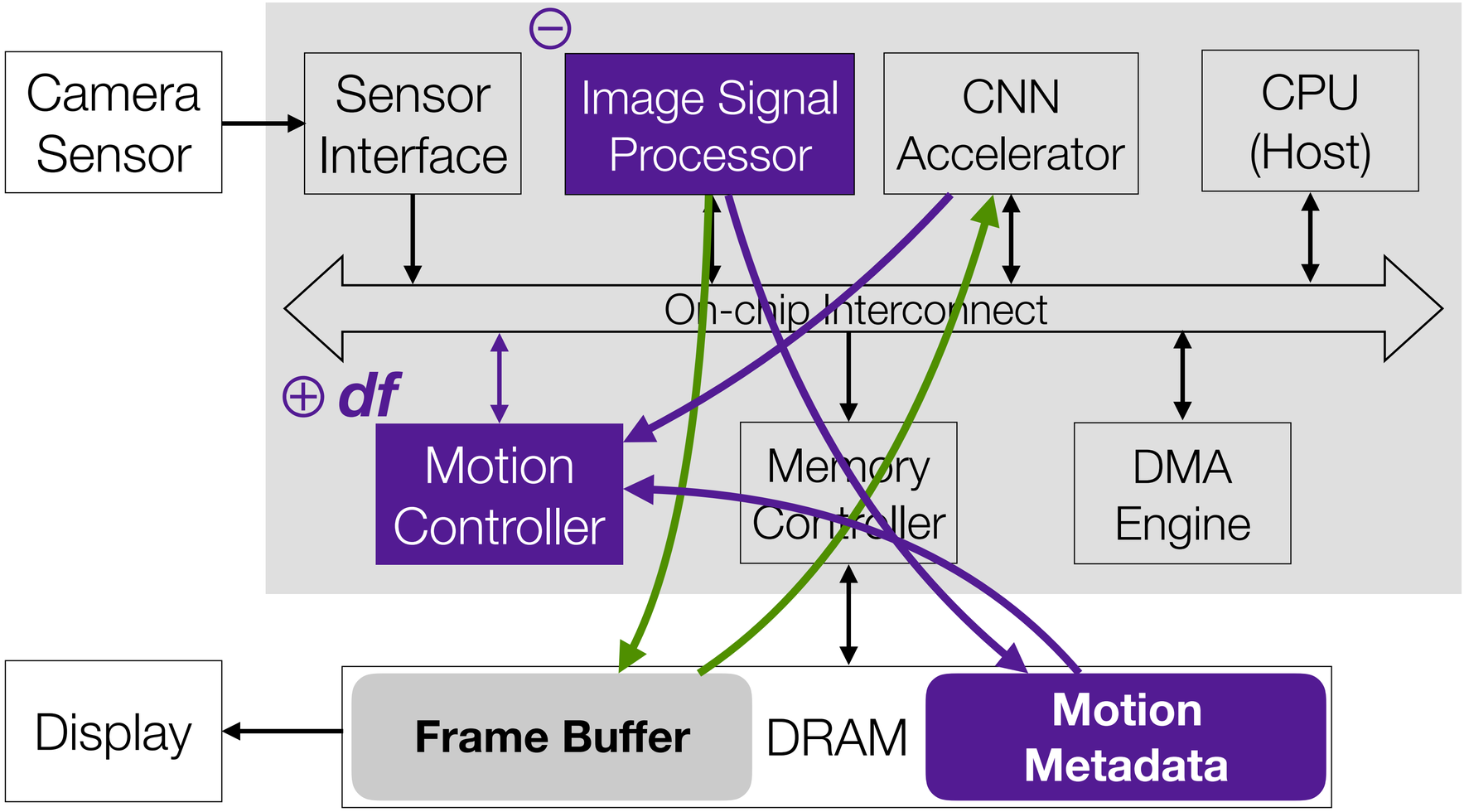
Hardware Architecture Support

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$



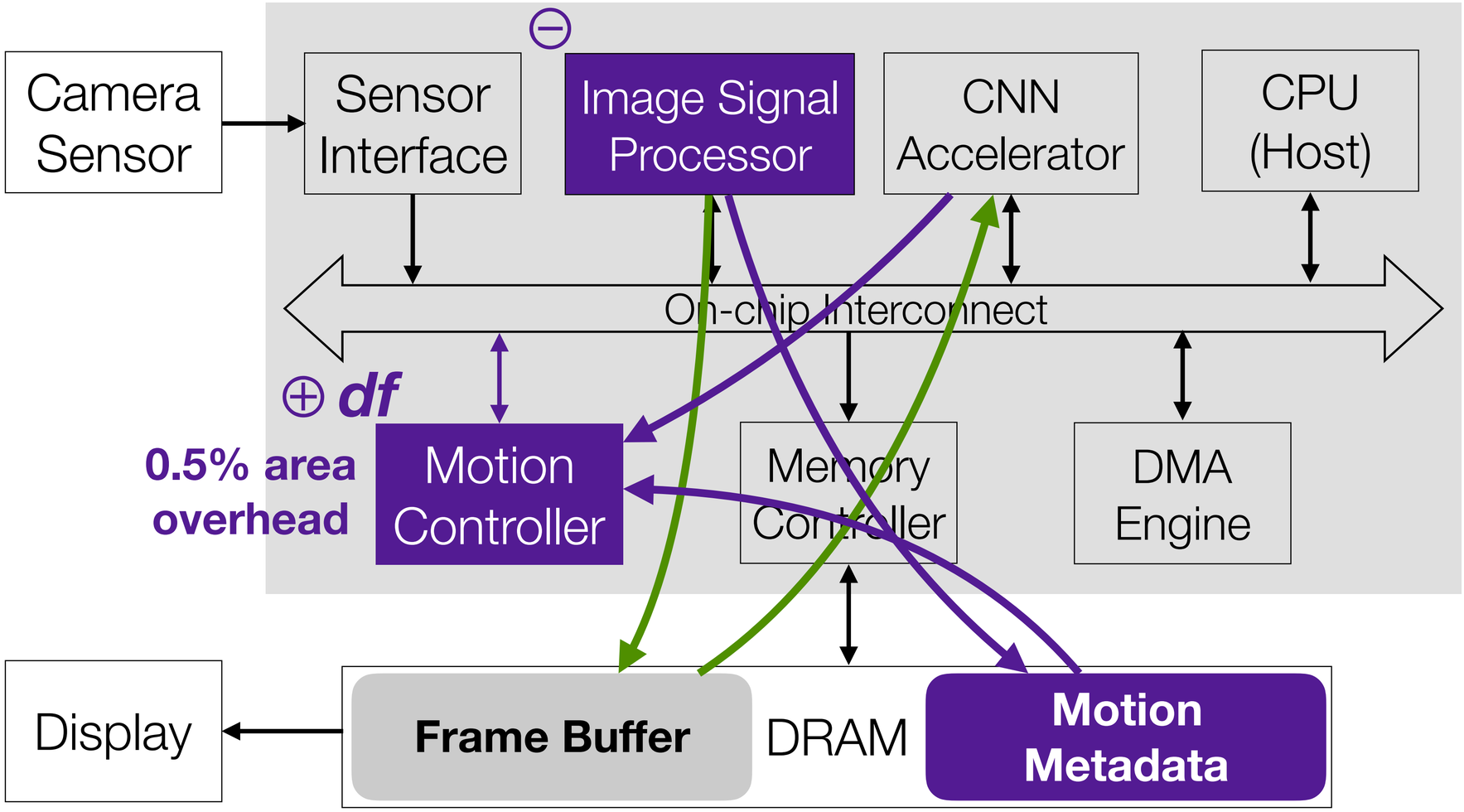
Hardware Architecture Support

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$



Hardware Architecture Support

$$y_{t+1} = df(x_{t+1} \ominus x_t) \oplus y_t$$



Evaluations

Vision algorithms: Object detection/tracking, **depth from stereo**

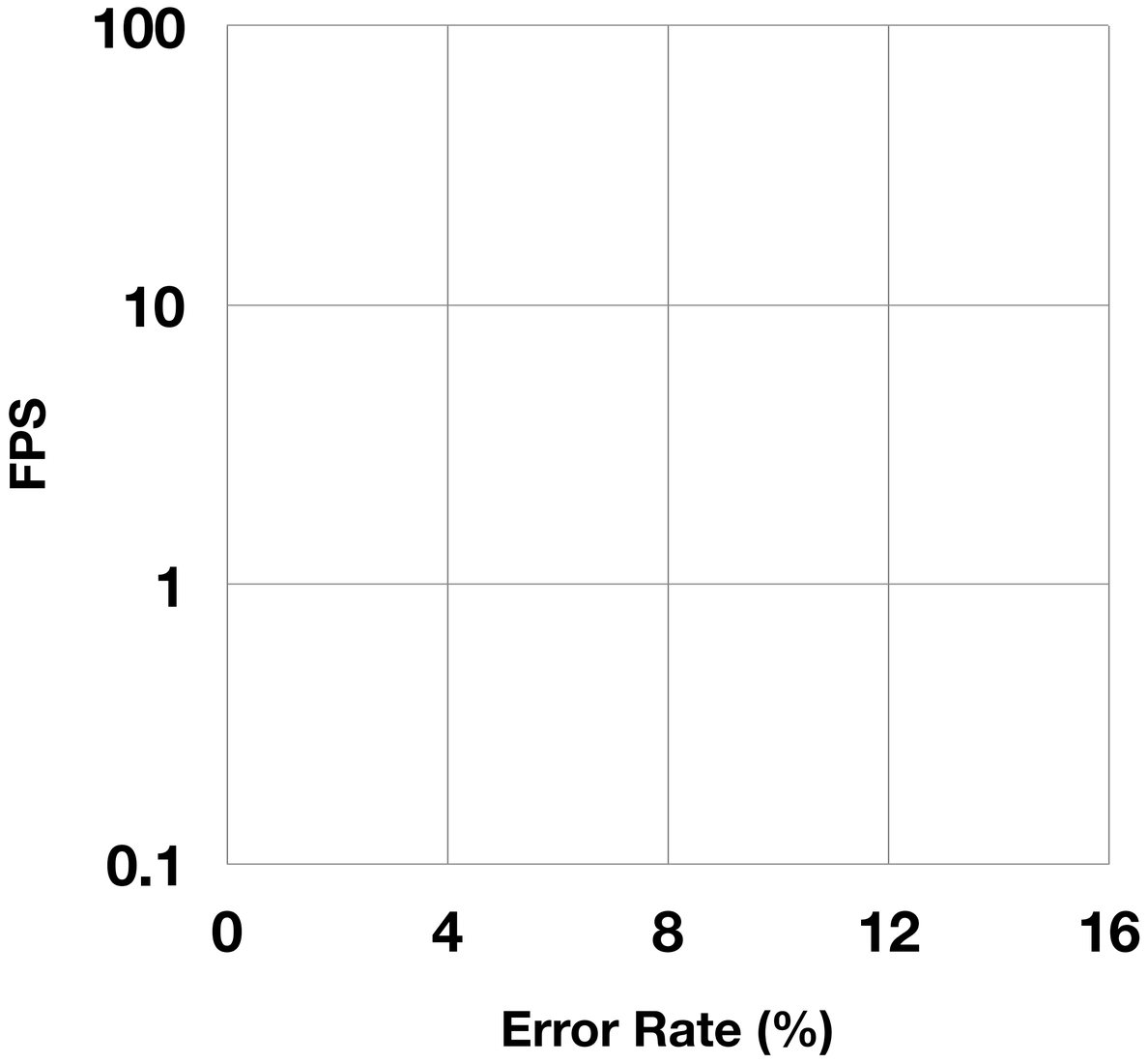
Baseline: State-of-the-art CNNs (DispNet, FlowNetC, GC-Net, and PSMNet)

Hardware: Develop RTL models, place and route in a 16nm process node

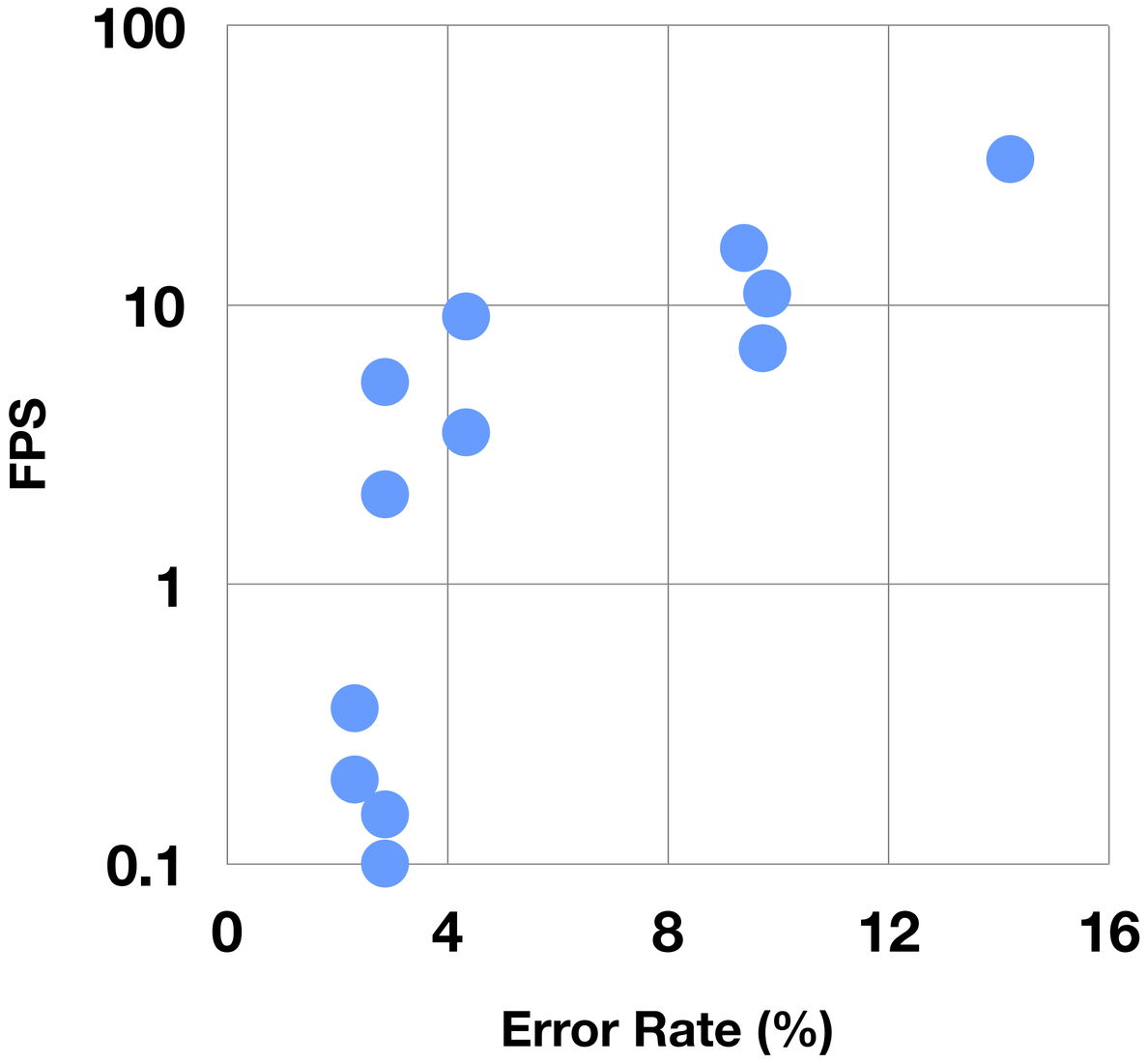
Software: `https://github.com/horizon-research/ism-algorithm`

Dataset: KITTI and SceneFlow datasets

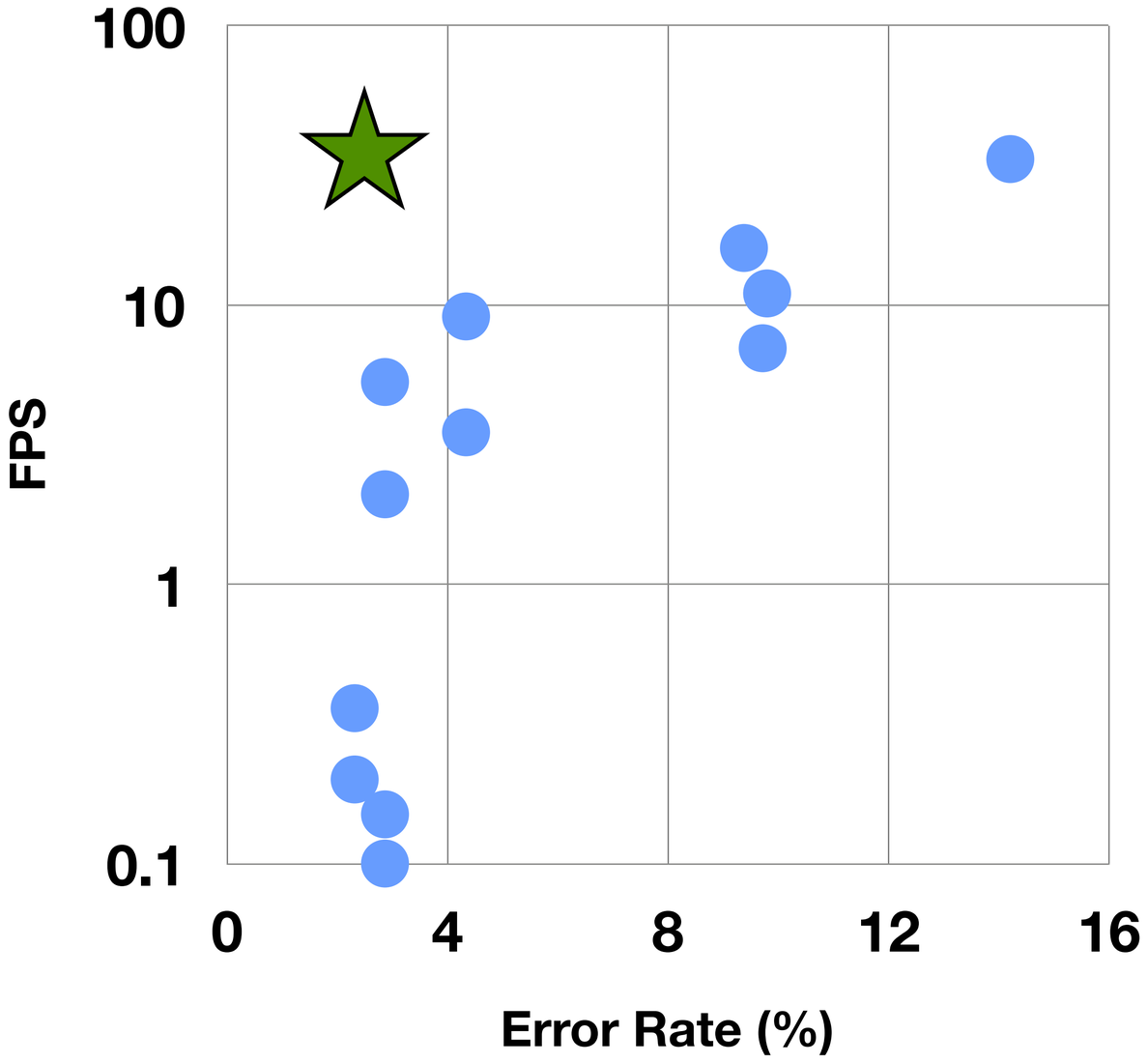
Results



Results



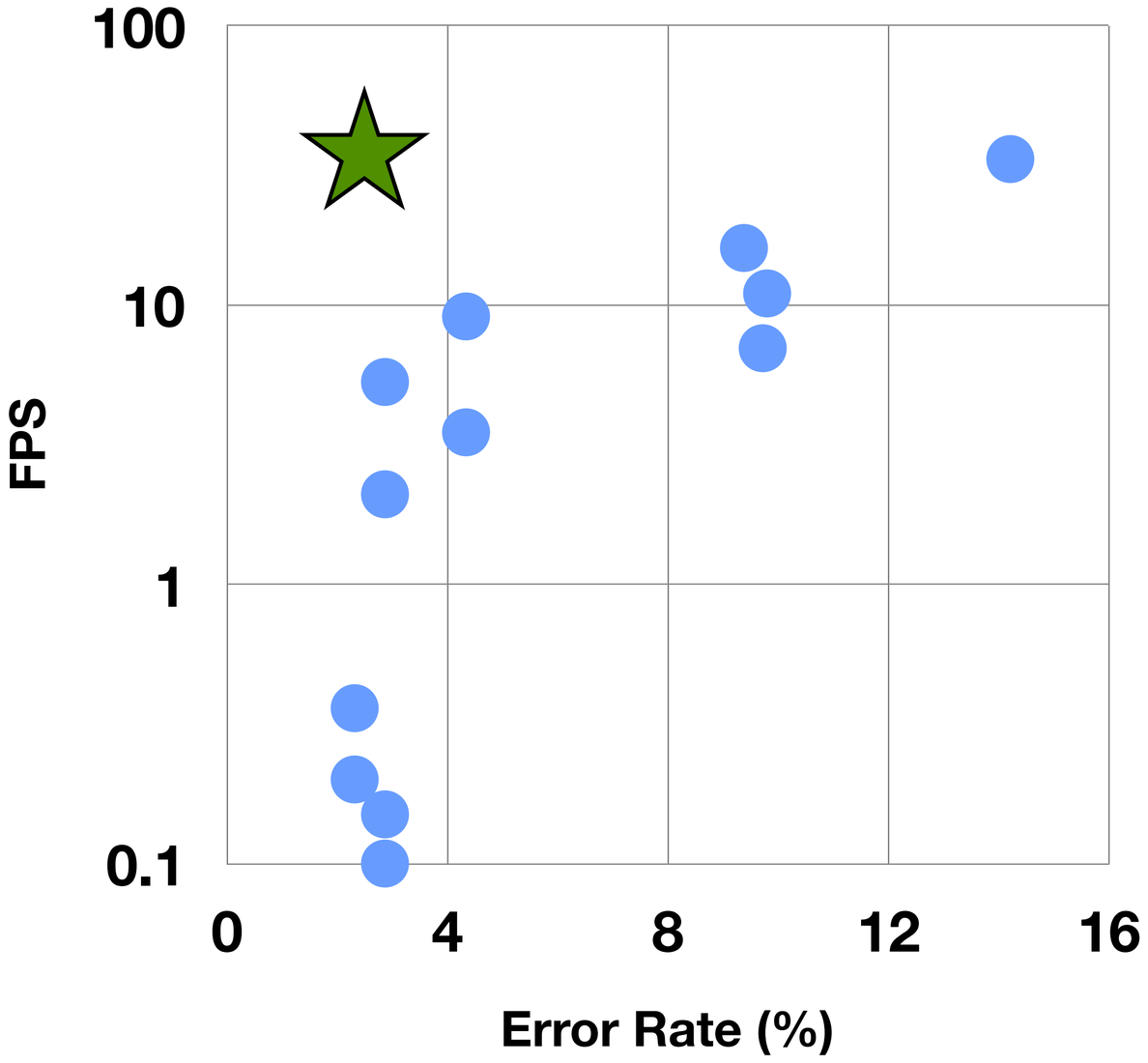
Results



Results

Compare to state-of-the-art CNN algorithms:

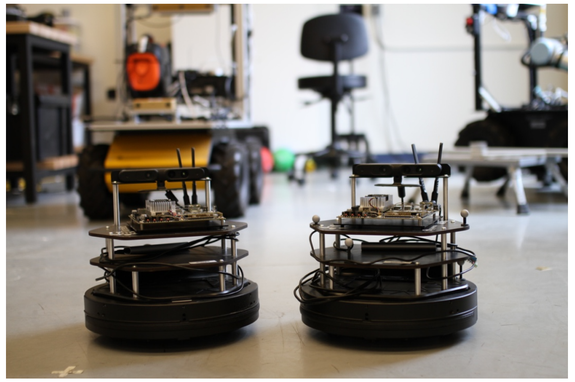
- 5.0 X speedup
- 6.7 X energy reduction
- 0.02% accuracy loss

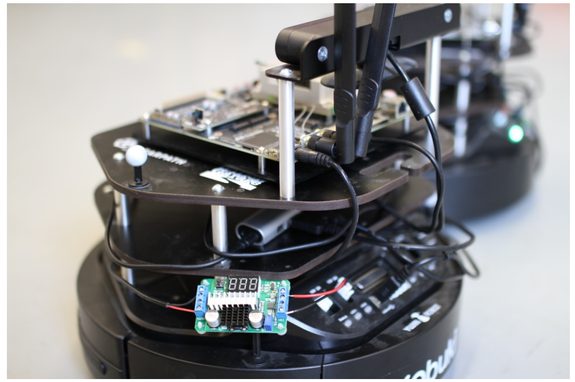


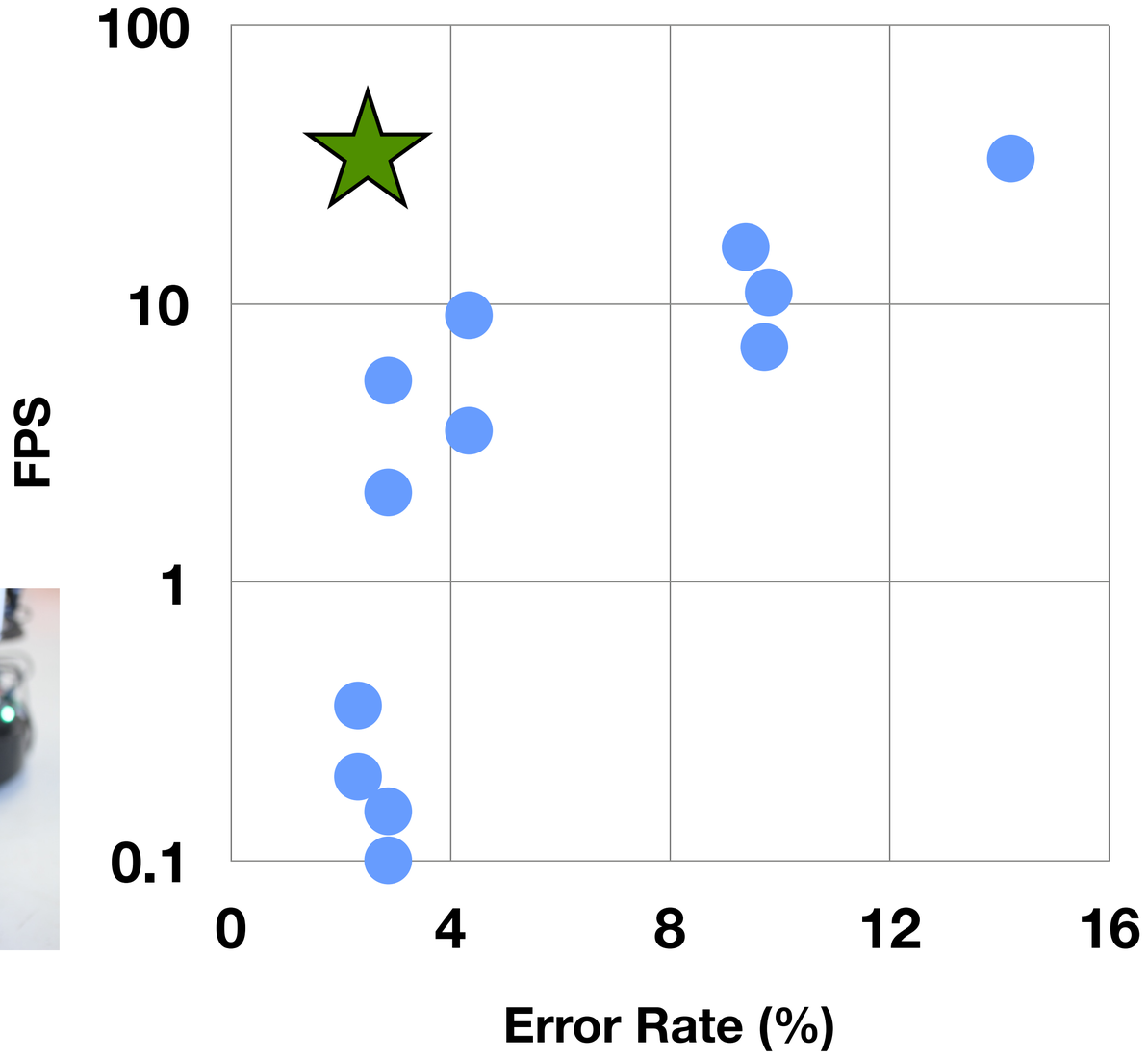
Results

Compare to state-of-the-art CNN algorithms:

- 5.0 X speedup
- 6.7 X energy reduction
- 0.02% accuracy loss

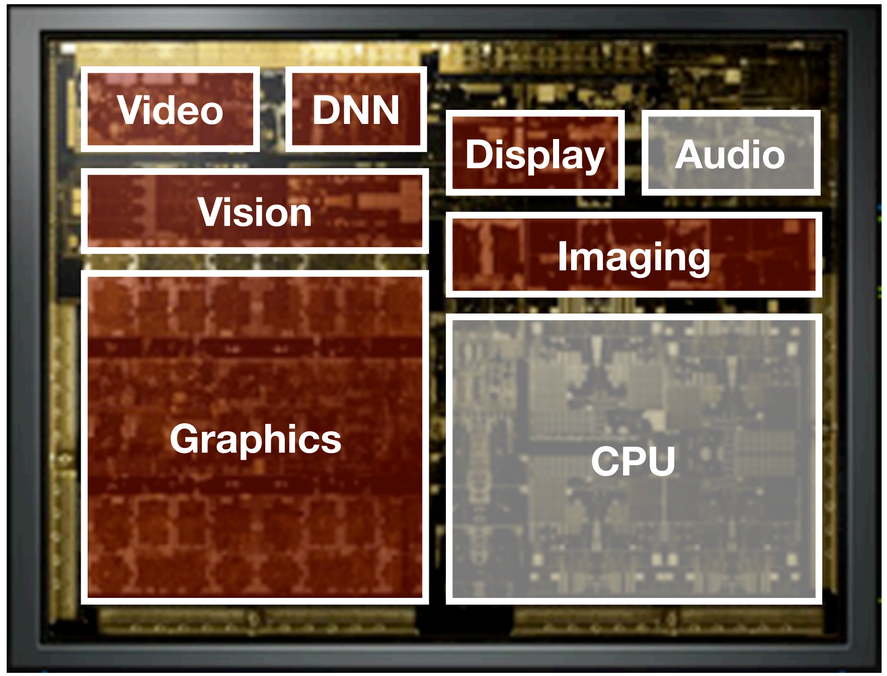






Co-Design Beyond Imaging and Vision

Domain	Tasks	⊖	df	⊕
Vision	Object Tracking/ Detection	Motion Vector	Averaging	Extrapolation
	Depth from Stereo	Motion Vector	Reconstructing Corresponding Pair	Triangulation
Imaging	Temporal Denoising	Motion Vector	—	Motion-Compensated Temporal Filtering
	Rolling Shutter Correction	Scanline-Level Motion Vector	—	Motion Compensation
Video Processing	Video Encoding	Motion Vector	—	Motion Compensation
	Video Stabalization	Egomotion Vector	—	Egomotion Compensatio
Display	Frame Upsampling	Motion Vector	Averaging	Interpolation
Graphics	Timewarp in VR	Egomotion Vector	—	Reprojection



Co-Design Beyond Imaging and Vision

Domain	Tasks	⊖	df	⊕
Vision	Object Tracking/ Detection	Motion Vector	Averaging	Extrapolation
	Depth from Stereo	Motion Vector	Reconstructing Corresponding Pair	Triangulation
Imaging	Temporal Denoising	Motion Vector	—	Motion-Compensated Temporal Filtering
	Rolling Shutter Correction	Scanline-Level Motion Vector	—	Motion Compensation
Video Processing	Video Encoding	Motion Vector	—	Motion Compensation
	Video Stabalization	Egomotion Vector	—	Egomotion Compensatio
Display	Frame Upsampling	Motion Vector	Averaging	Interpolation
Graphics	Timewarp in VR	Egomotion Vector	—	Reprojection

Visual Applications

Programming Abstractions

⊖ ⊕ **df**

Compiler

Programable Hardware Substrate

⊖ **df** ⊕ ...

Addressing the world's most pressing issues requires systems that can generate and interpret massive visual data in real time using extremely low power

Medicine & Healthcare

Environmental Sustainability

Cultural Preservation

https://med.stanford.edu/news/all-news/2017/09/virtual-reality-alleviates-pain-anxiety-for-pediatric-patients.html

https://www.producthunt.com/posts/rumii

https://idealog.co.nz/tech/2018/07/new-form-cultural-preservation-how-one-new-zealander-using-vr-transport-people-tombs-egypt

Addressing the world's most pressing issues requires systems that can generate and interpret massive visual data in real time using extremely low power.

The only way we will get there: software-hardware co-design (duh!). How should we go about co-design?

Addressing the world's most pressing issues requires systems that can generate and interpret massive visual data in real time using extremely low power.

The only way we will get there: software-hardware co-design (duh!). How should we go about co-design?

Our view: break away from the traditional vertical, domain-specific mindset, and embrace the horizontal, cross-domain mindset, which means:

Addressing the world's most pressing issues requires systems that can generate and interpret massive visual data in real time using extremely low power.

The only way we will get there: software-hardware co-design (duh!). How should we go about co-design?

Our view: break away from the traditional vertical, domain-specific mindset, and embrace the horizontal, cross-domain mindset, which means:

- Co-design across different visual computing domains

Addressing the world's most pressing issues requires systems that can generate and interpret massive visual data in real time using extremely low power.

The only way we will get there: software-hardware co-design (duh!). How should we go about co-design?

Our view: break away from the traditional vertical, domain-specific mindset, and embrace the horizontal, cross-domain mindset, which means:

- Co-design across different visual computing domains
- Co-design across the computing and non-computing (e.g., optics) domains

HAJIM
SCHOOL OF ENGINEERING
& APPLIED SCIENCES
COMPUTER SCIENCE

The Next Quintillion Pixels:

Architecting Next-Generation Mobile Visual Computing Systems

More at: http://horizon-lab.org

Yu Feng

Amir Taherin

Qiuyue Sun

Anand Samajdar
(@ GaTech)

Paul Whatmough
(@ Arm Research)

Matt Mattina
(@ Arm Research)