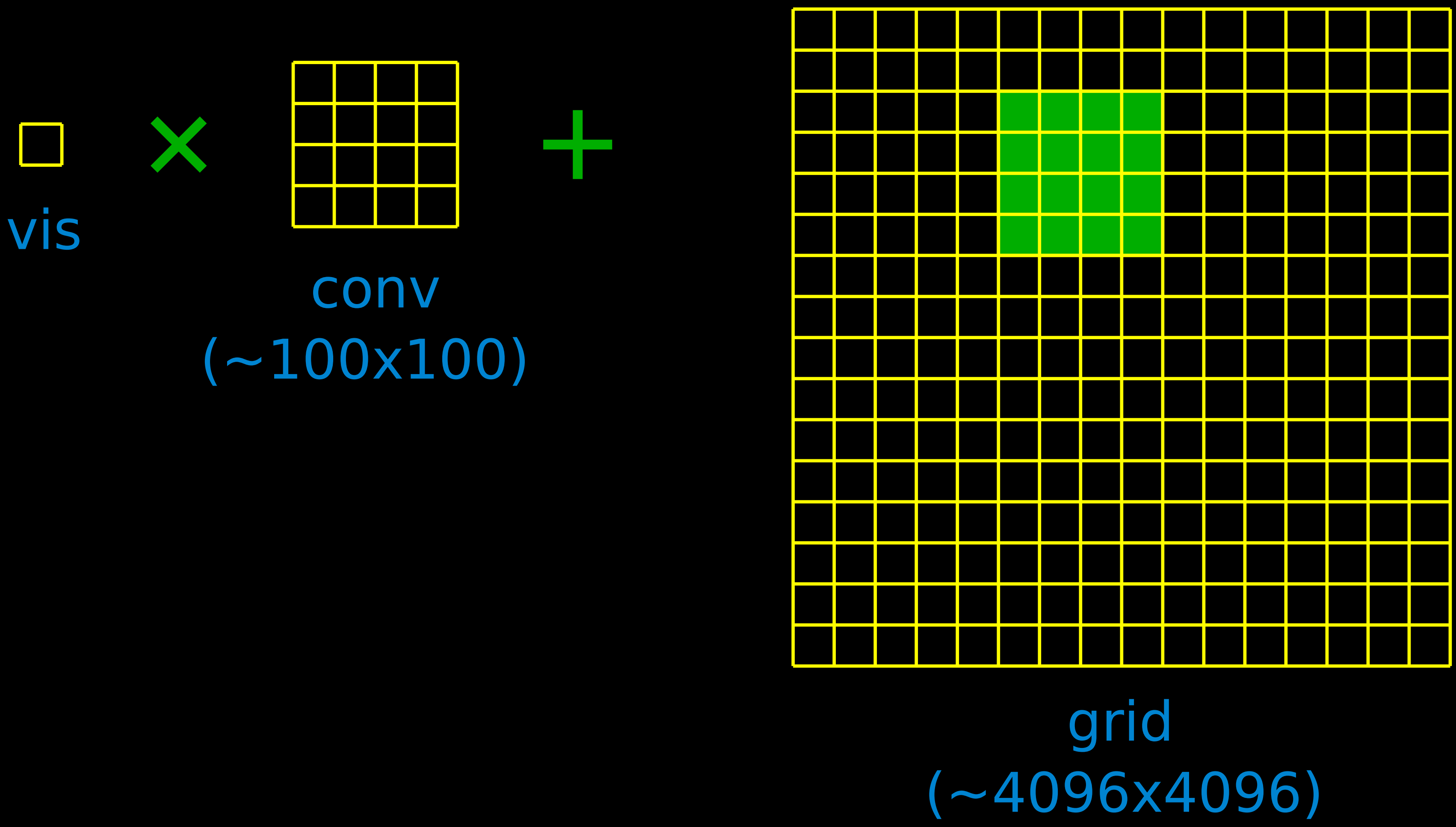
Fast W-Projection Gridding on GPUs

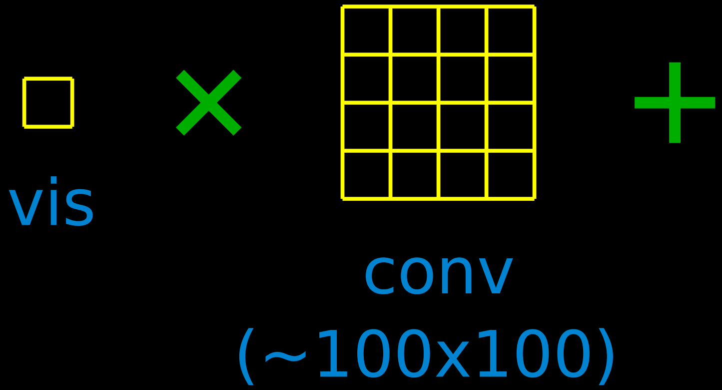
John W. Romein

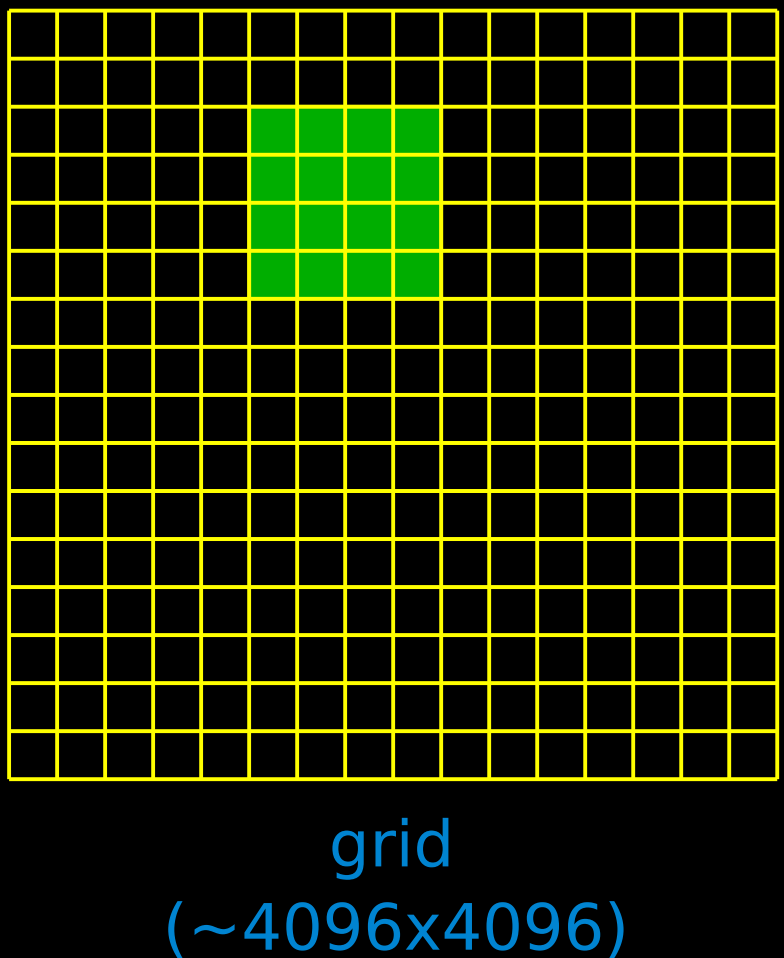
Stichting ASTRON (Netherlands Institute for Radio Astronomy)
Dwingeloo, the Netherlands

Gridding



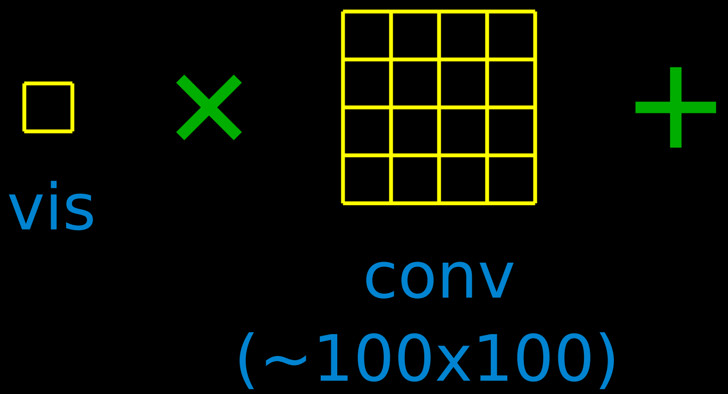
Two Problems

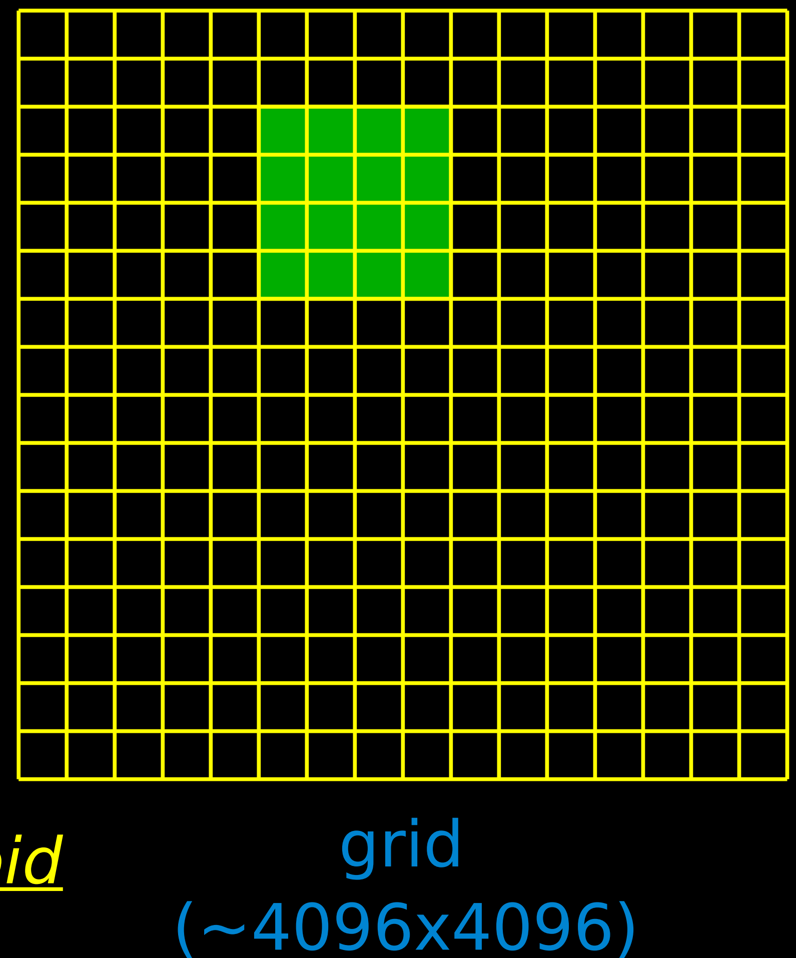




1. lots of FLOPS
2. add to memory: slow!

Two Solutions





1. lots of FLOPS ➜ *use GPUs*
2. add to memory: slow! ➜ *avoid*

Outline

- GPU introduction
- W-projection gridding on GPUs
- performance results

GPUs

GPUs

- powerful compute device
- highly parallel
- device memory
  - “limited” bandwidth

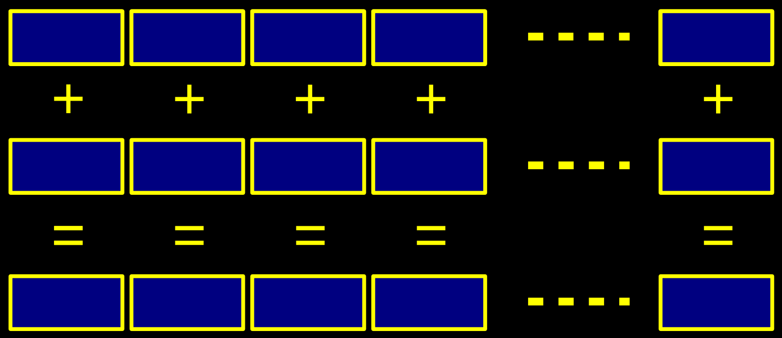
	CPU *(E5620)*	GPU *(GTX 580)*	GPU / CPU
cores	4	512	128
threads	8	16,384	2,048
vector length	4		
GFLOPS	76.8	1,581	20.6
memory BW (GB/s)	25.6	192.4	7.52
TDP (W)	80	244	3.05

GPU Compute Model

- model:
  - move data CPU → GPU
  - run kernel on GPU
  - move result GPU → CPU
- PCIe often bottleneck
- overlap computations and communication

GPU Features

- core hierarchy: 16 multi-processors (SMs) of 32 cores
  - SMs independent
  - cores in SM cooperate
    - SIMD
    - coalescing
    - latency hiding: ≤32 threads/core



- textures
  - efficient 2D/3D caching
  - interpolation (indexed by floating point number)

GPU Memories

slow

fast

- host*
  - on CPU
- device*
  - on GPU
- texture*
  - 2D/3D caching, 2D/3D interpolation, normalization, ...
- constant*
- shared
  - 48KB per SM
- register

(* = cached)

GPU Programming

- CPU code in C/C++
- GPU code in CUDA or OpenCL

GPU Languages

- OpenCL
  - Nvidia, AMD, ...
  - CPU side: C horrible, C++ very pleasant
- CUDA
  - Nvidia only
  - better support for latest GPU features
  - 2%~20% faster
  - matured more

CUDA Example

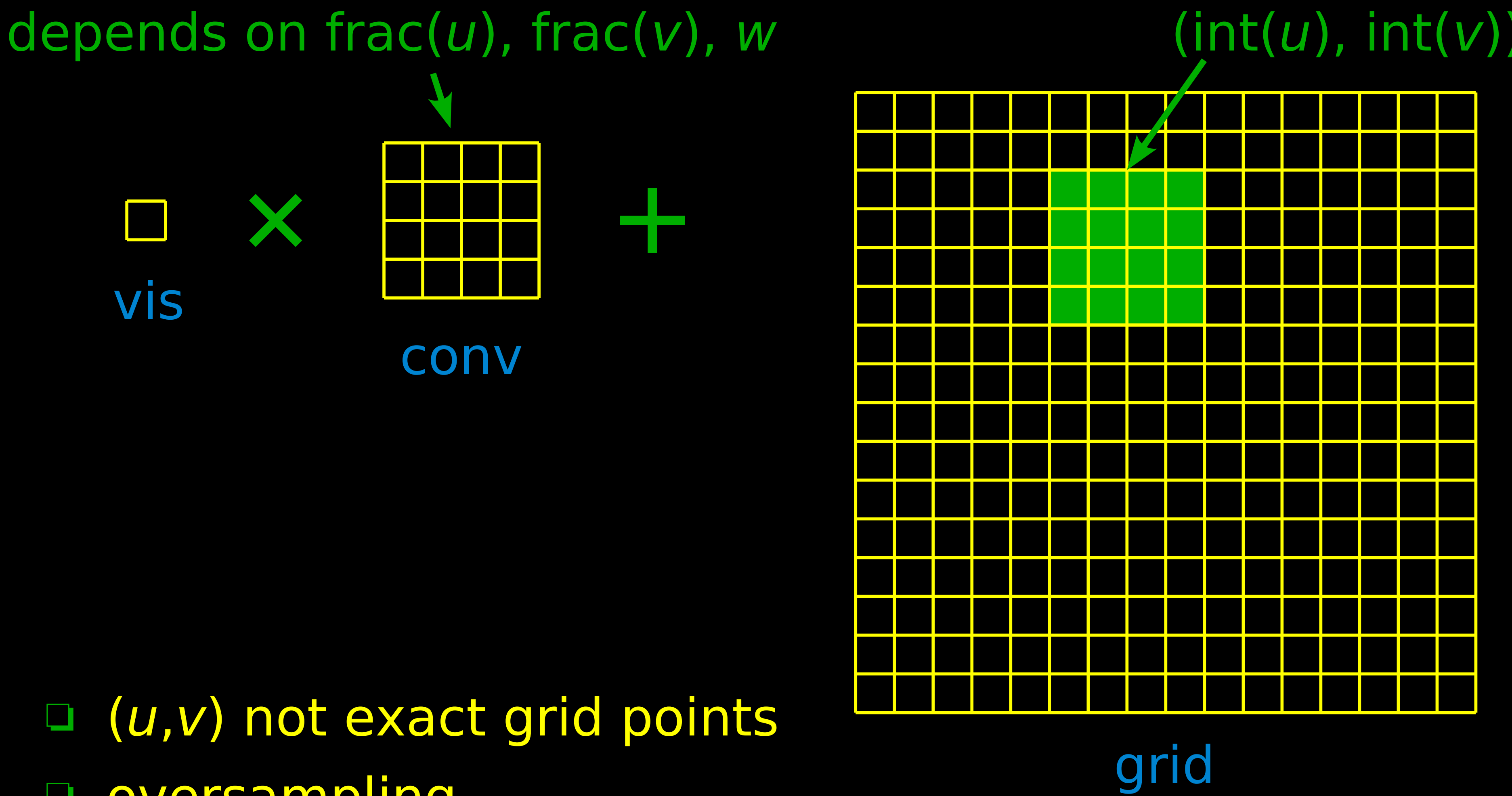
```
__device__ float array[1024];

__global__ void zero_array()
{
   array[threadIdx.x] = 0;
}

int main()
{
   zero_array<<<1, 1024>>>();
   return 0;
}
```

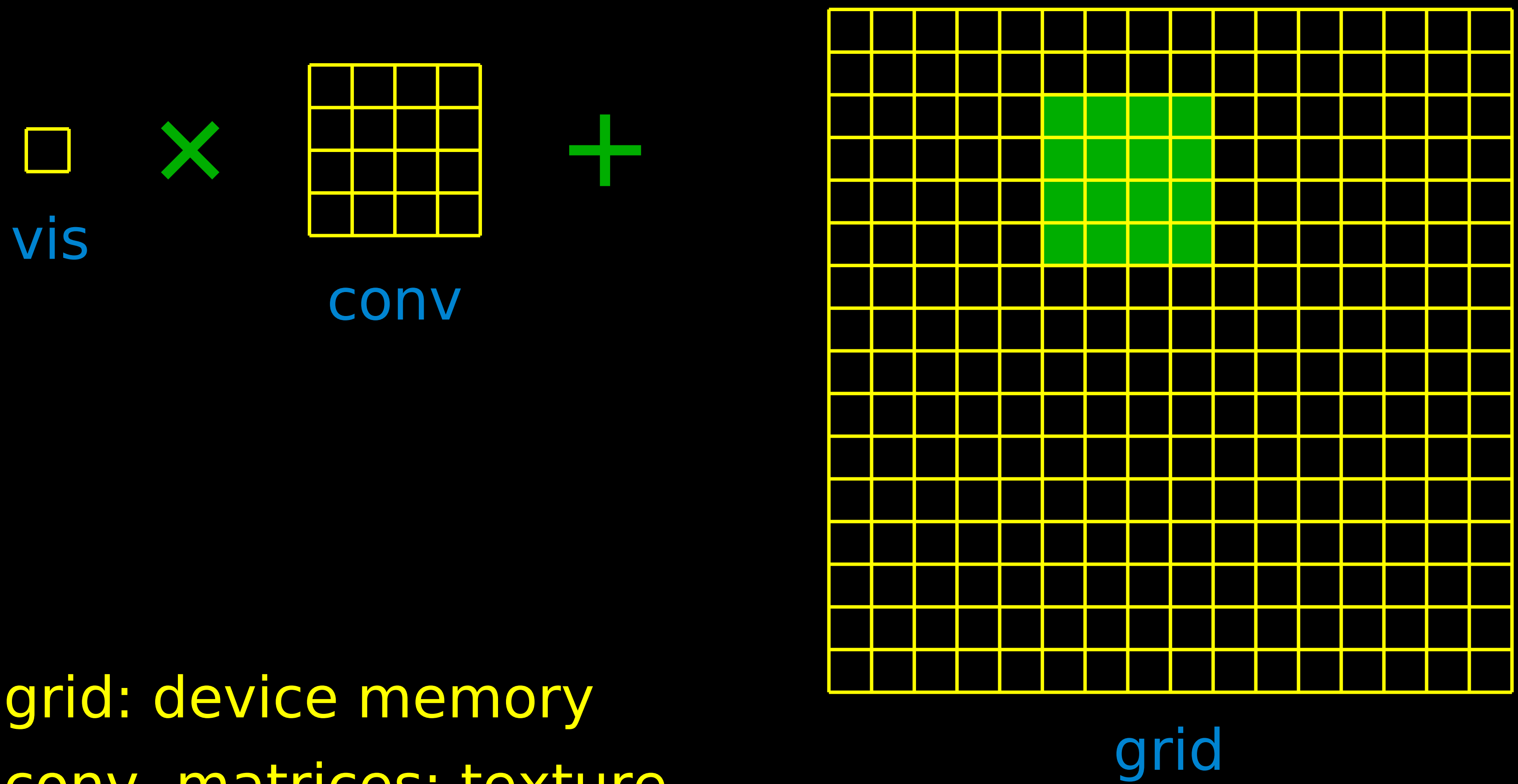
Back To Gridding

W-Projection Gridding



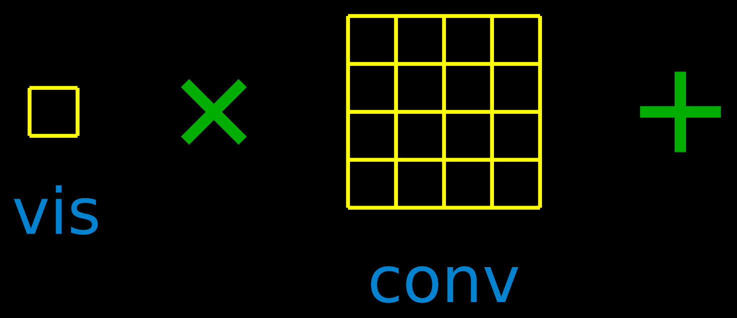
- (*u*,*v*) not exact grid points
- oversampling
  - choose most appropriate conv. matrix

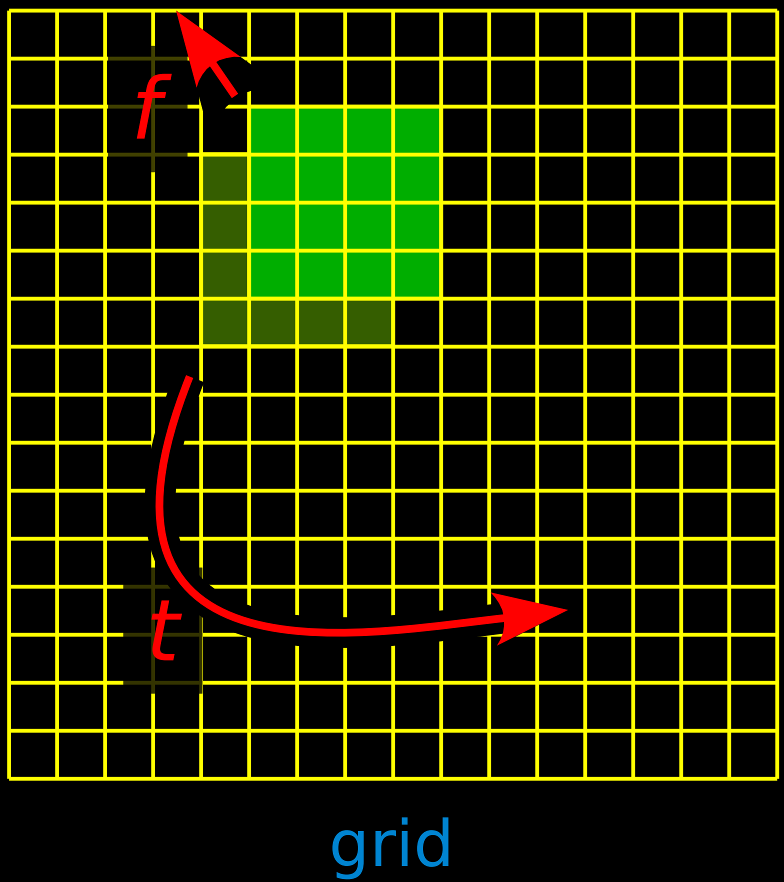
Where Is The Data?



- grid: device memory
- conv. matrices: texture
- vis. + (*u*,*v*,*w*): shared memory

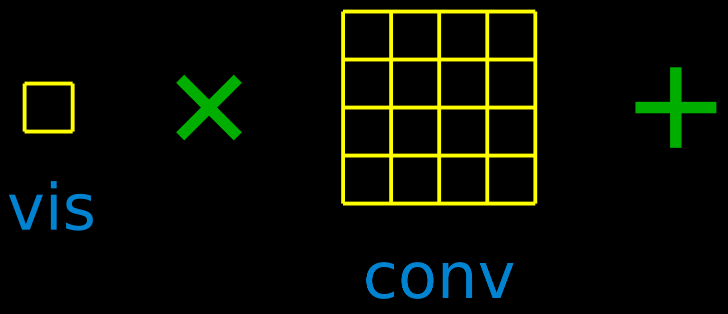
Placement Movement

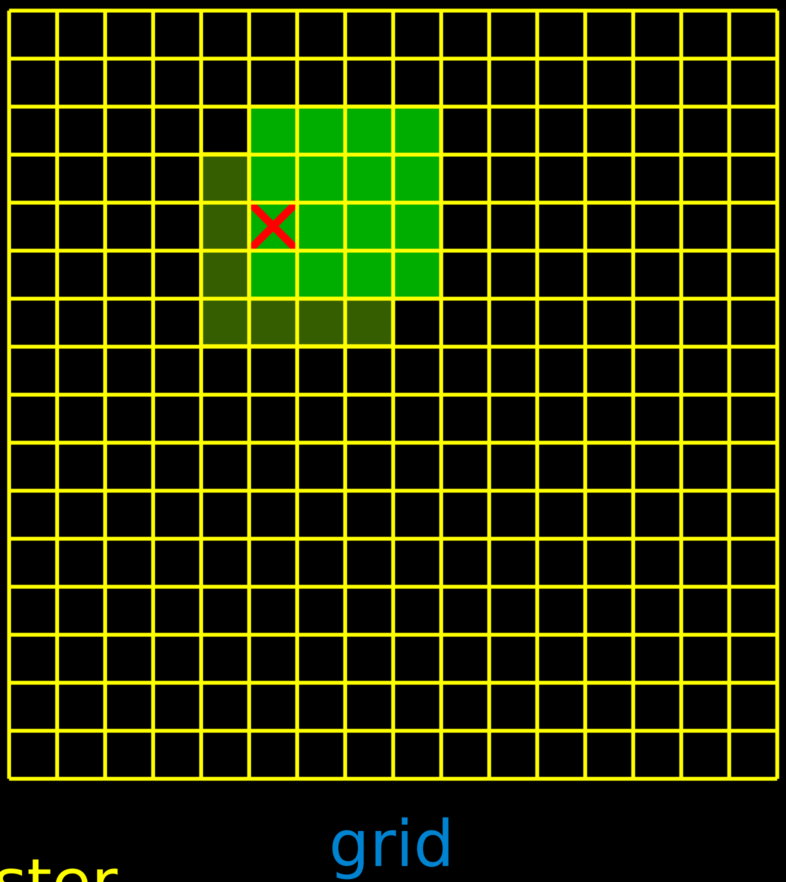




- per baseline:
  - (*u*,*v*,*w*) changes slowly
  - grid locality

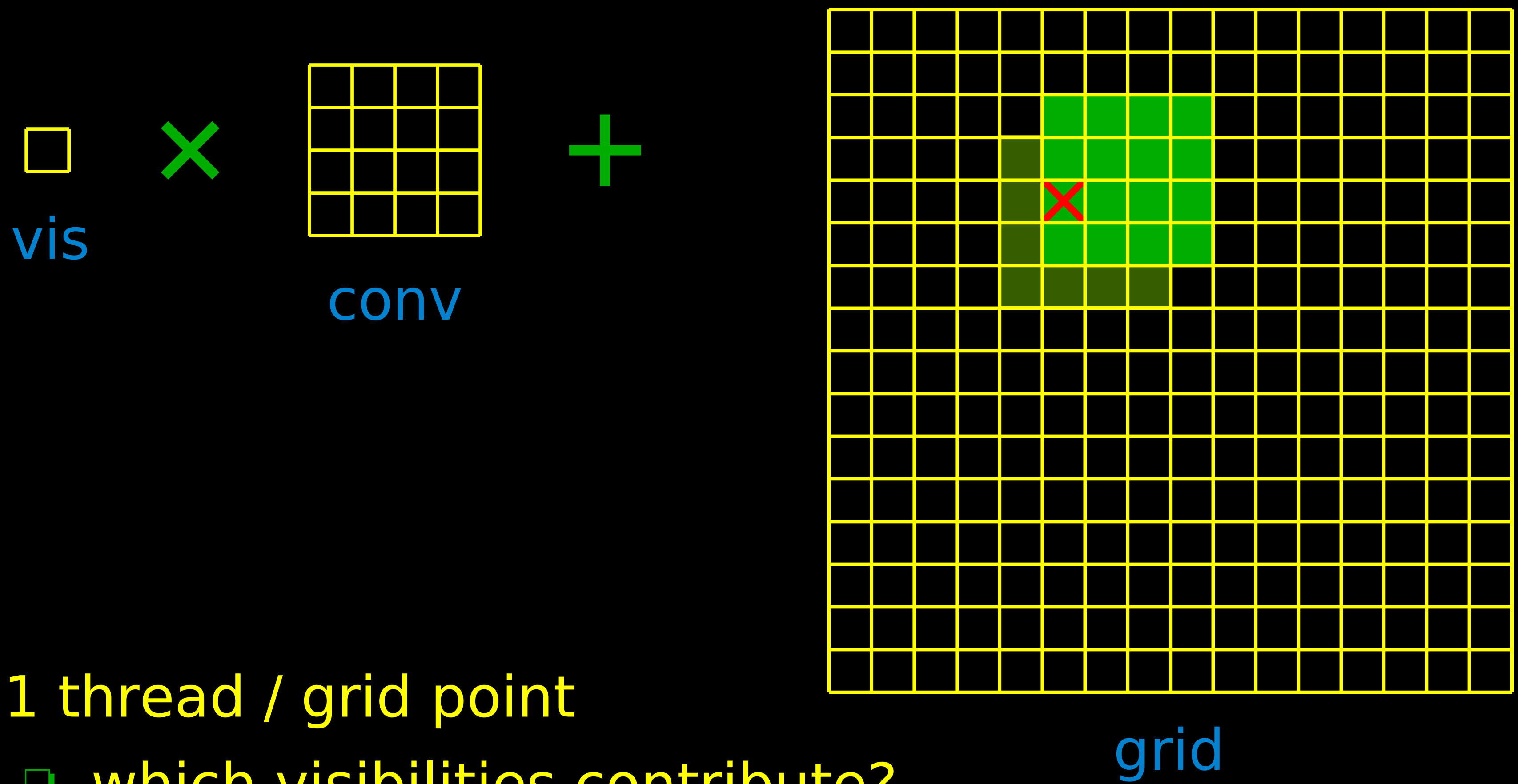
Use Locality





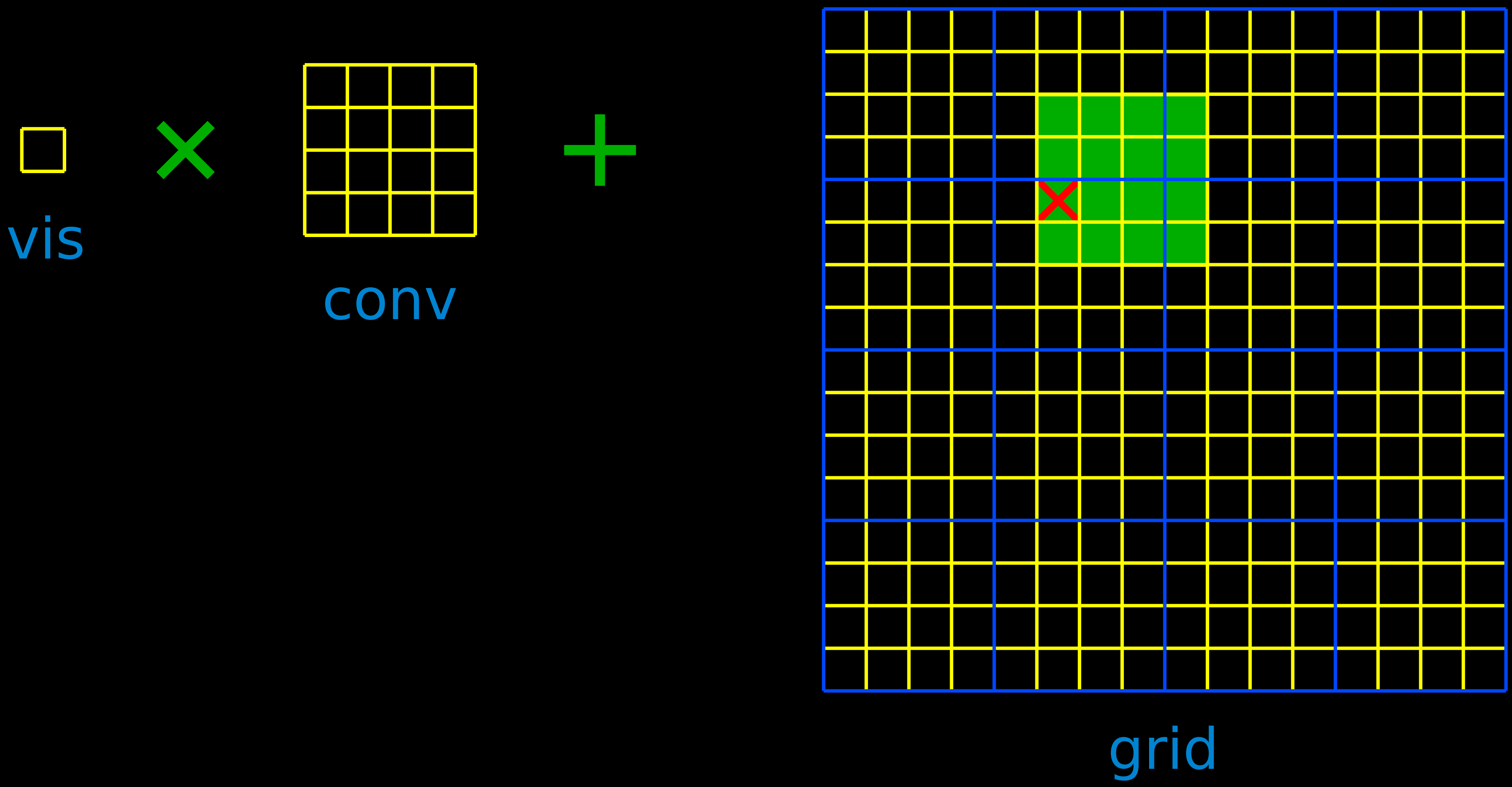
- reduce #memory accesses
- **X**: one thread
- accumulate additions in register
- until conv. matrix slides off

But How ???



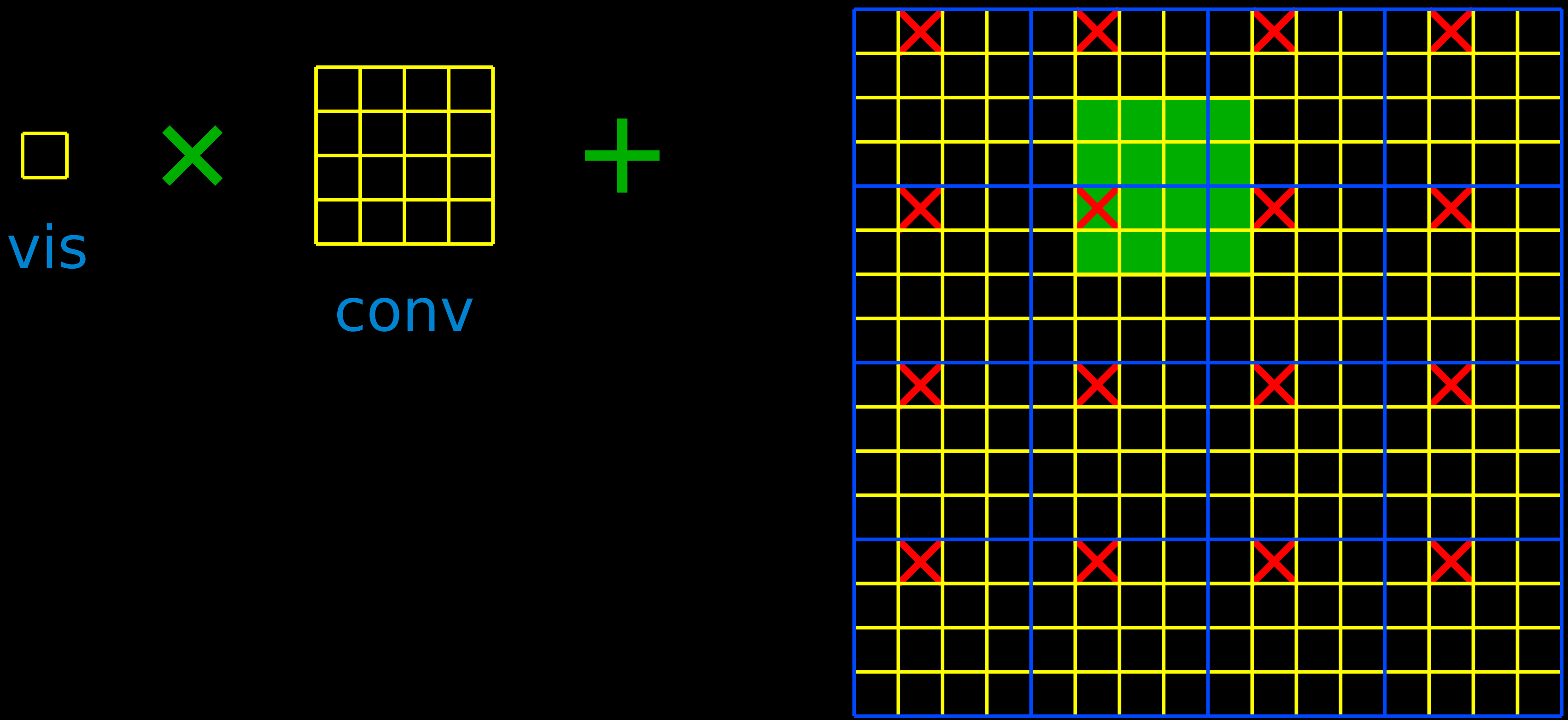
- 1 thread / grid point
  - which visibilities contribute?
  - severe load imbalance

An Unintuitive Approach



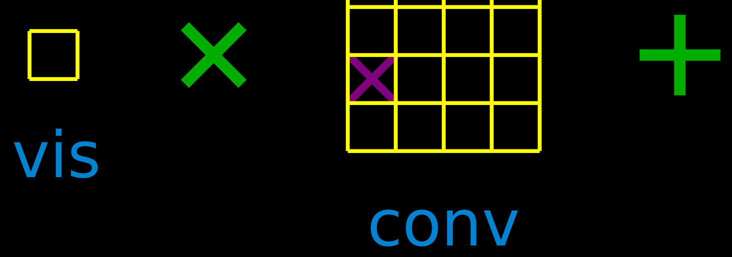
- conceptual blocks of conv. matrix size

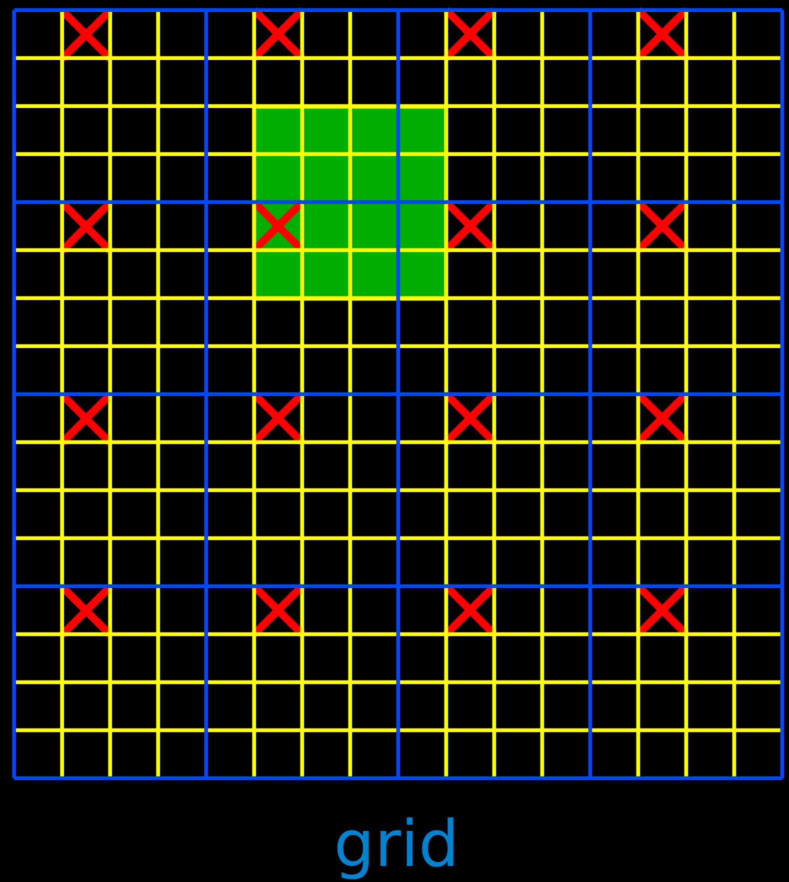
An Unintuitive Approach



- 1 thread monitors all **X**
- at any time: conv. matrix covers <u>1</u> **X**!!!

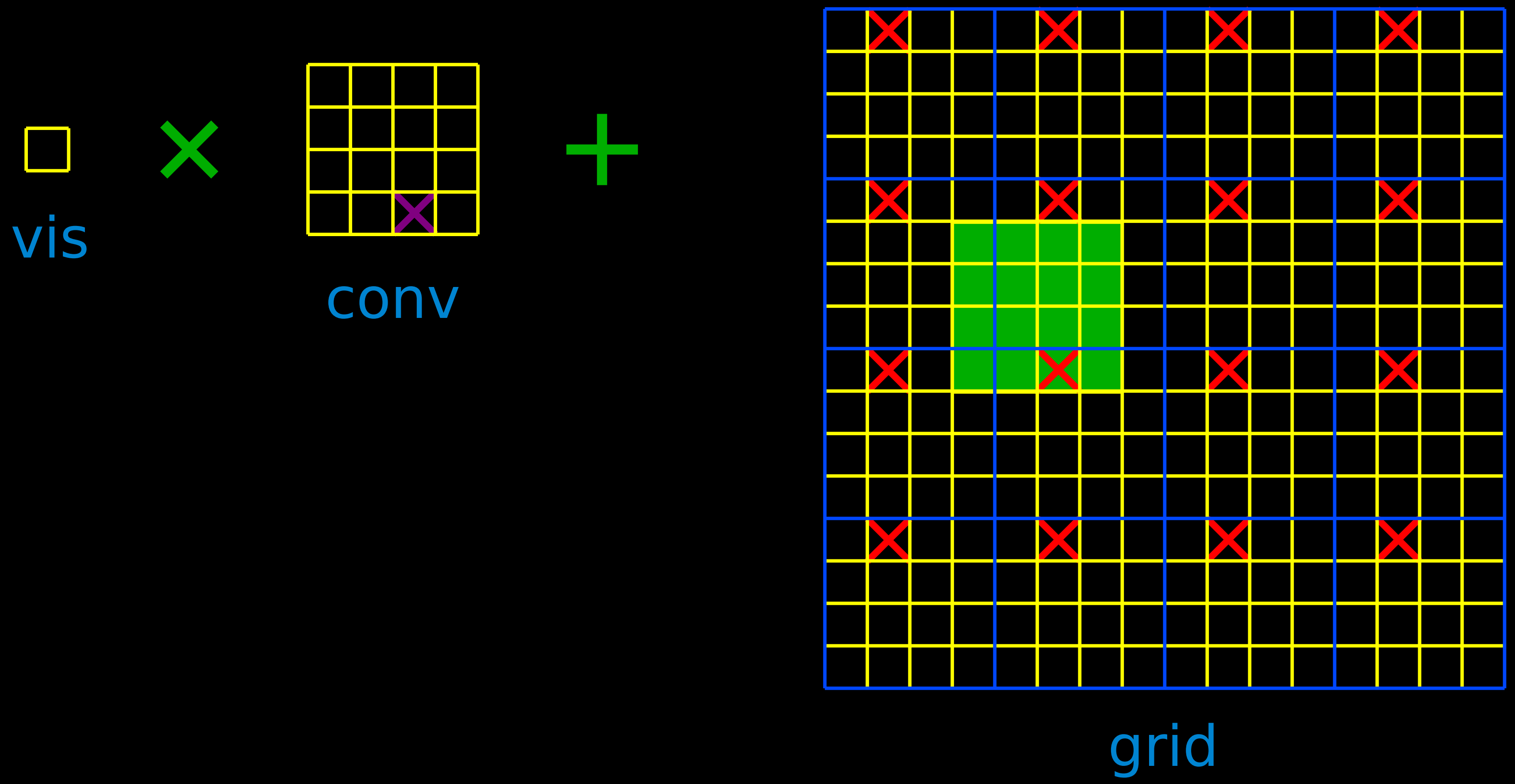
An Unintuitive Approach





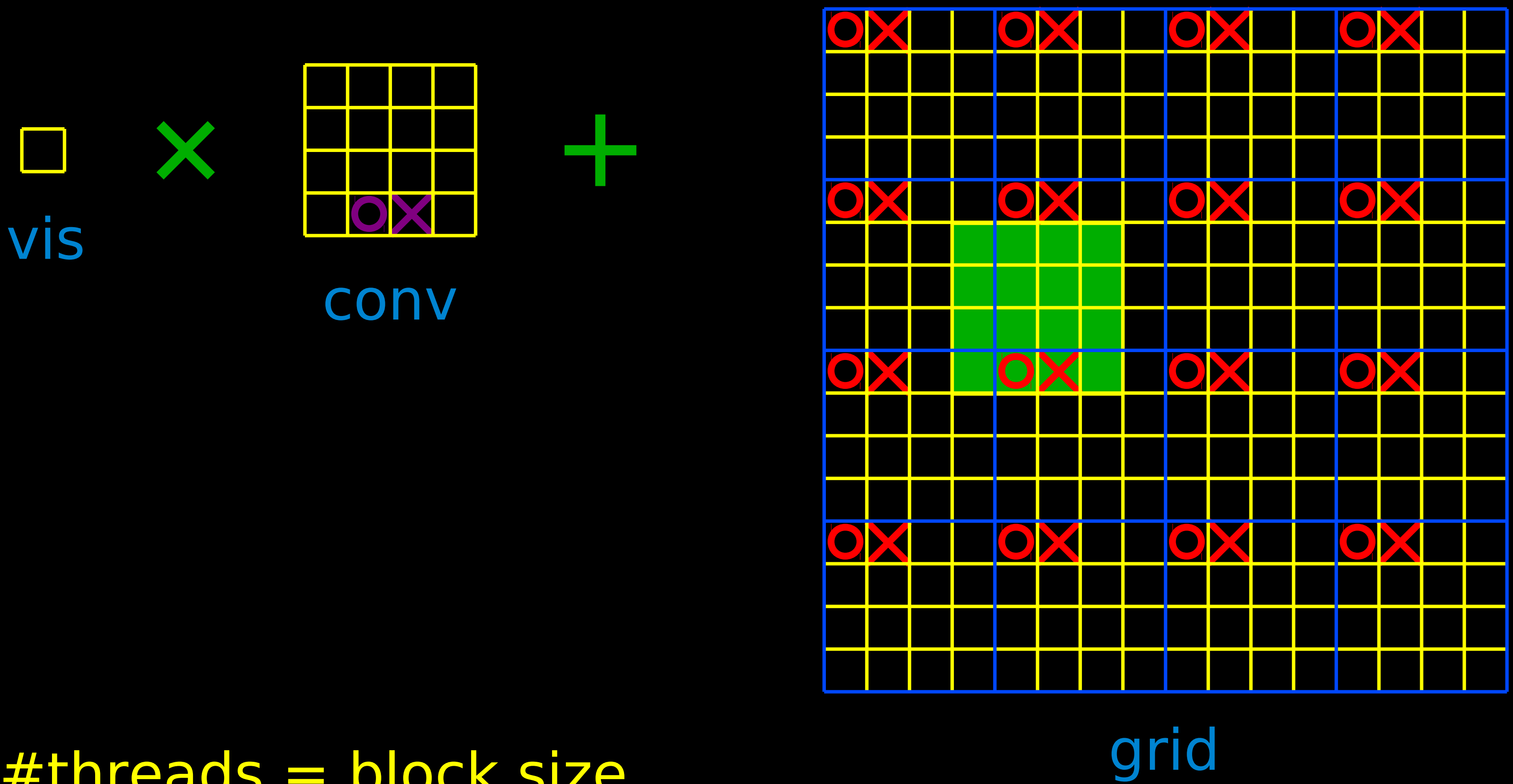
- thread computes current:
  - X grid point
  - X conv. matrix entry

An Unintuitive Approach



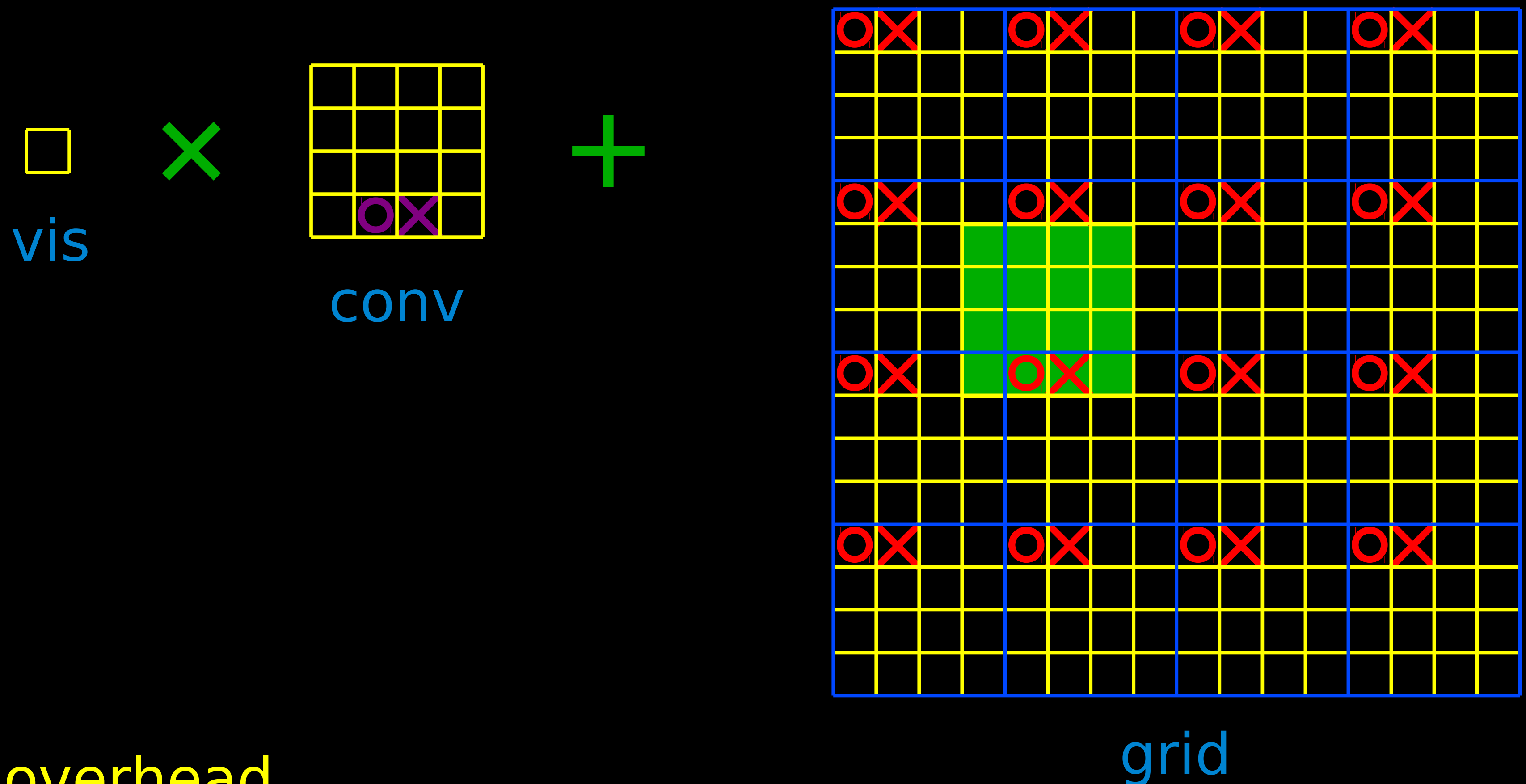
- (atomically) adds data if moved to another **X**

An Unintuitive Approach



- #threads = block size
  - too many threads → do in parts

(Dis)Advantages



- ☹ overhead
- ☺ < 1% grid-point memory updates

Work Distribution

- baselines: spread over SMs
- times: threads in SM
- frequencies: threads in SM
- polarizations: single thread

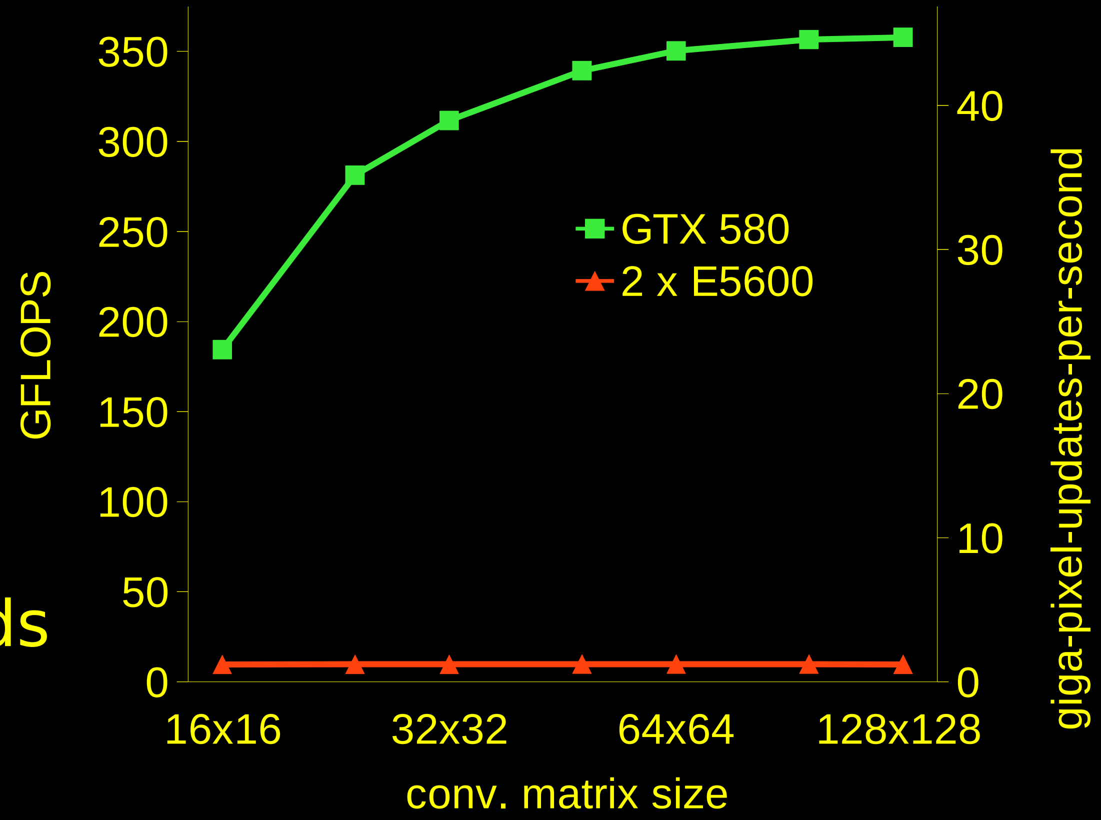
Performance Measurements

Performance Tests Setup

#stations	44
#channels	16
integration time	10 s
observation time	6 h
conv. matrix size	≤ 128x128
oversampling	8x8
#W-planes	128
grid size	4096x4096

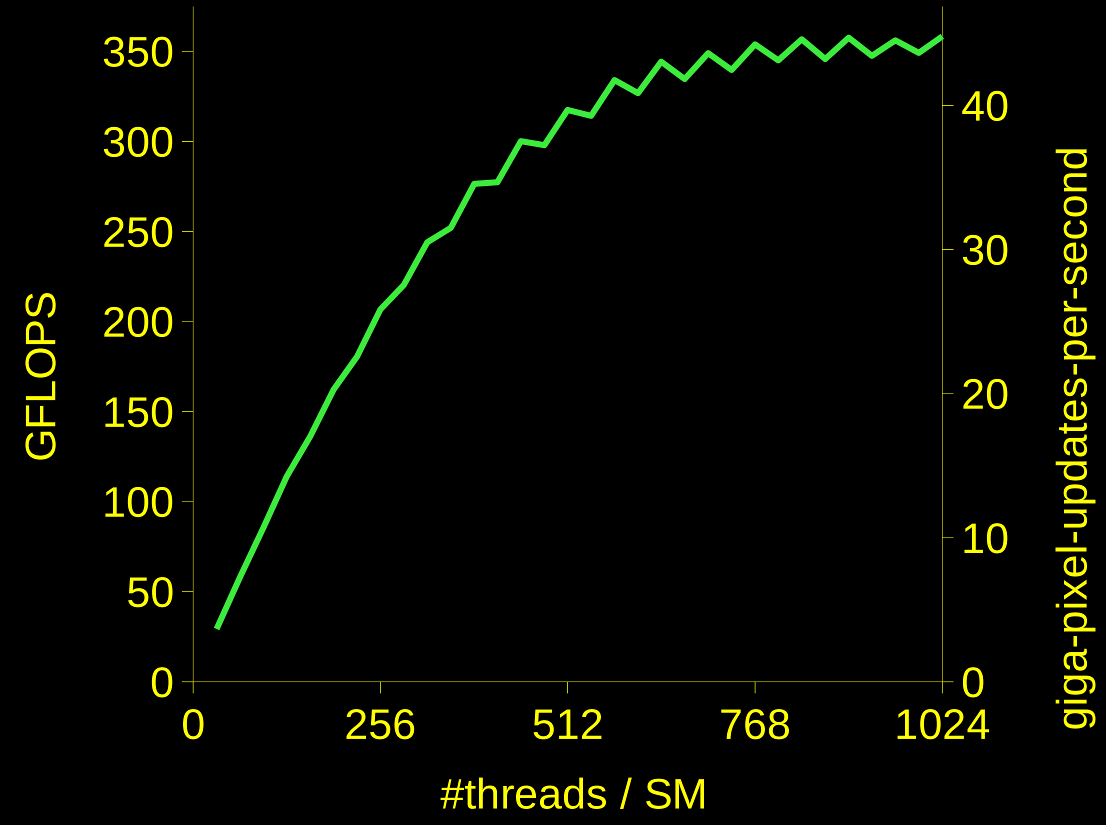
- (*u*,*v*,*w*) from real observation (6 hour)

CUDA Performance



- ≤37x dual CPU
- 16x16:
  - insufficient #threads
  - more atomic adds
  - (>1 baseline/SM)
- independent of #W-planes
- PCIe bus utilization: 17%–0.5%

#Threads



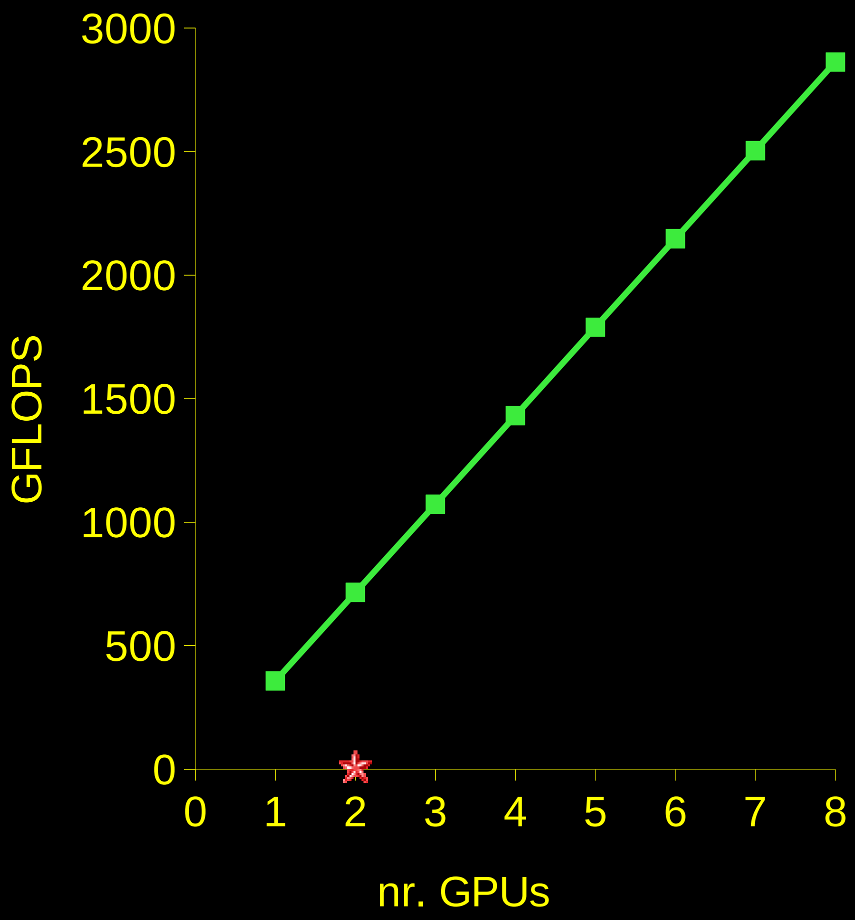
- 128x128 conv. matrix

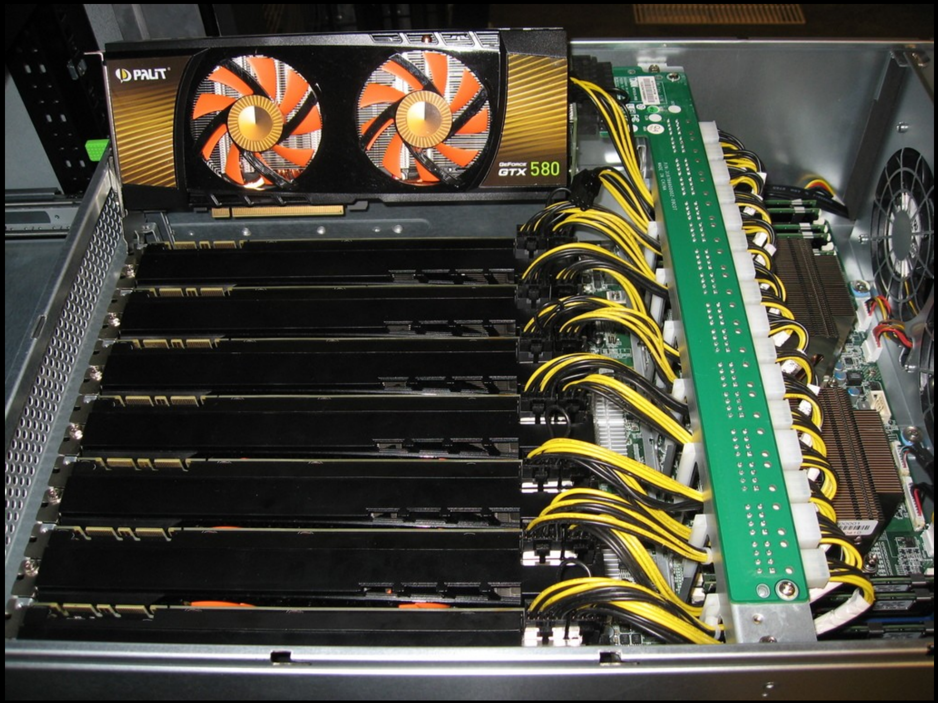
OpenCL Performance

- language bit restrictive
  - no 1D textures
  - no atomic add ➜ use atomic cmpxchg
- Nvidia GTX 580
  - 18% slower than CUDA
  - multi-GPU/host-threads issues
- AMD HD 6970
  - limited grid size (2048 x 2048)
  - 13-163x slower than GTX 580!
    - atomic ops slow

Multi-GPU Scaling

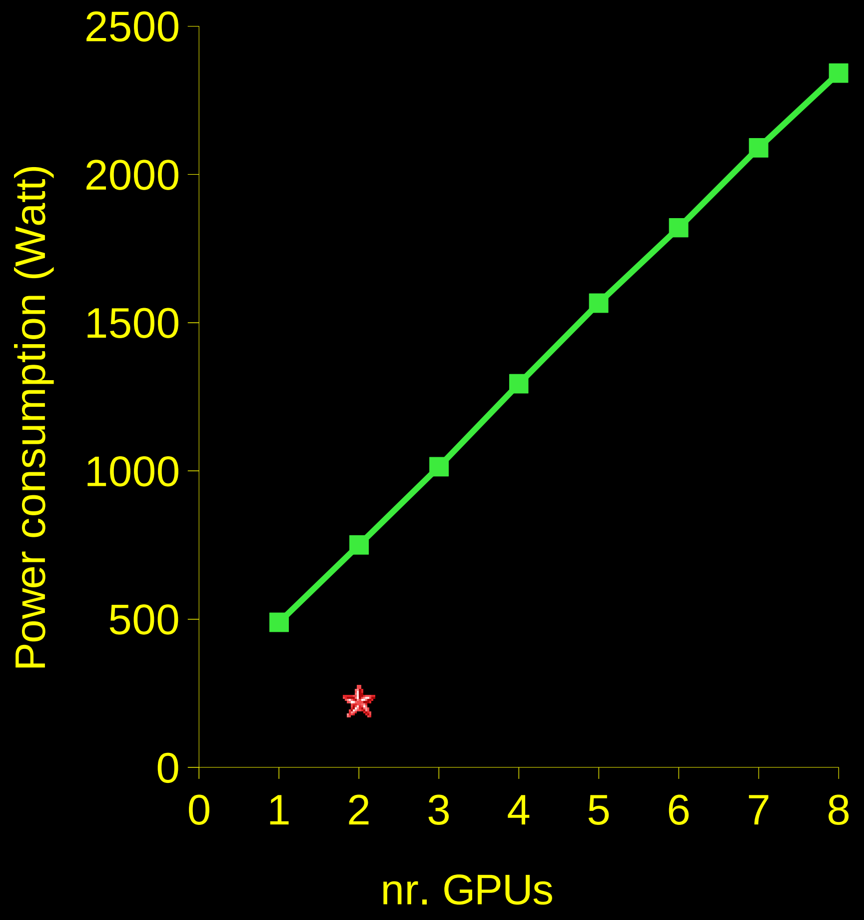
- eight Nvidia GTX 580s

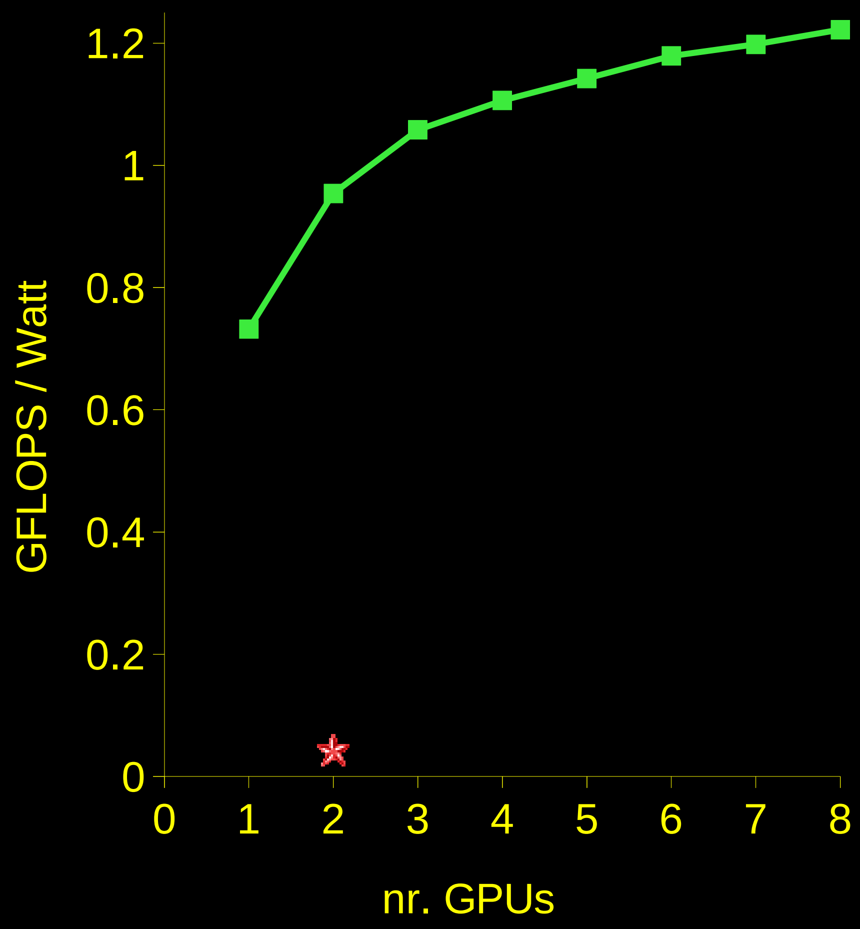




- 131,072 threads!
- scales perfectly
- 296x faster than dual CPU

Green Computing





- 28x more energy efficient than dual CPU

Comparison With Other GPU Gridders

- van Amesfoort et. al. [CF'09]
  - private grid per block ➜ very small grids
  - 3.5~6.5 x (compensated for faster hardware)
- MWA gridder (Edgar et. al. [CPC'11])
  - search visibilities that potentially add to grid point
  - 6.1~8.0 x
- Humphreys & Cornwell [SKA memo 132, '11]
  - adds directly to grid in memory
  - 8.5~10.3 x

Future Work

- work in progress
  - performance counters
- use hardware interpolation instead of oversampling/W-planes
- LOFAR gridder
  - combine with A-projection

Conclusions

- efficient GPU gridding algorithm
  - minimize memory accesses
- CUDA more mature than OpenCL
- 6~10x faster than other gridders
- 37x faster than dual CPU
  - scales perfectly on 8 GPUs
  - energy efficient