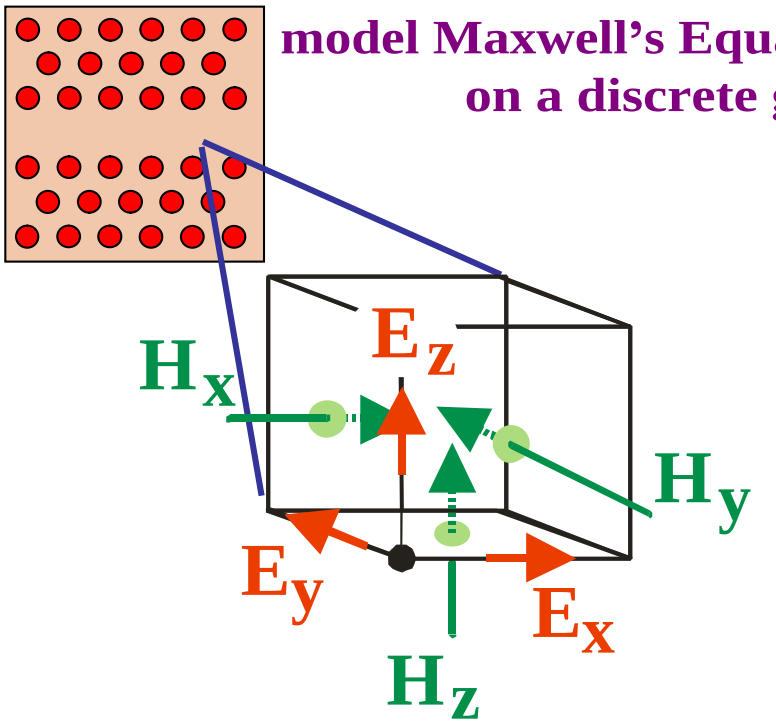
Using Grids to Model Nanoscopic Devices

Geoffrey W. Burr

IBM Almaden Research Center
San Jose, California 95120

- **Finite-difference time-domain (FDTD)**
  - **directly simulate Maxwell's equations for nanophotonics**
  - **as a motivation for...**
- **'Kitchen-sink' grid computing**
  - ***OptimalGrid* – IBM middleware for grid computing**
  - **MPI - Message-Passing Interface**
- **Conclusions**

What is FDTD?



model Maxwell's Equations on a discrete grid...

$$\mu \frac{\partial \vec{H}}{\partial t} = -\nabla \times \vec{E}$$

$$\epsilon_0 \epsilon_r \frac{\partial \vec{E}}{\partial t} = \nabla \times \vec{H} - \vec{J}_{source} - \sigma \vec{E}$$

$$E_y \mathrel{+}= \frac{\Delta t}{\varepsilon_r \varepsilon_0} \left(\frac{H_z - H_z^{\text{previous cell}}}{\Delta x} \right)$$

Why would we need it?

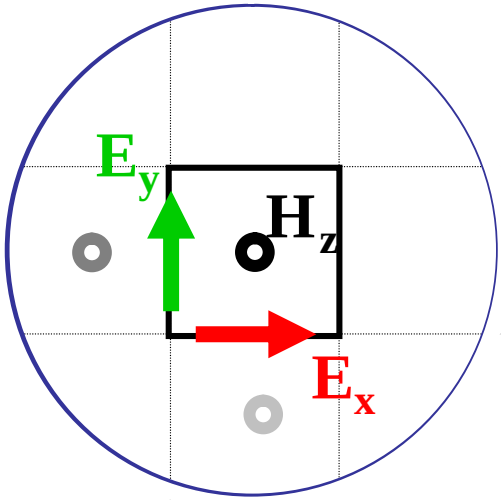
- *design-by-nanofabrication* is too expensive...
  - need accurate modeling/simulation tools
- FDTD is rigorous yet flexible (general-purpose tool)

‘Leapfrog’ update

1) at time t: Update E fields everywhere using spatial derivatives of H

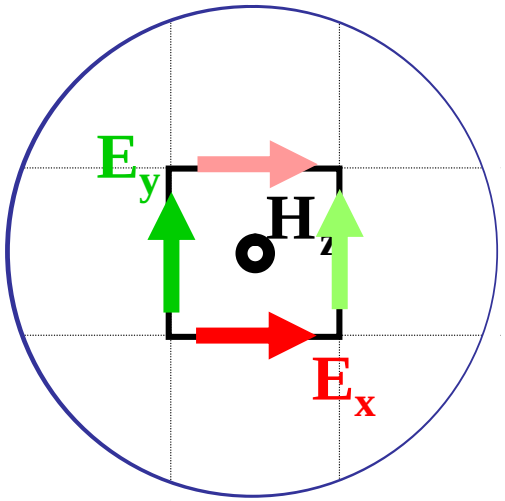
$$E_x \mathrel{+}= \frac{\Delta t}{\Delta y\, \varepsilon_r \varepsilon_0}\left(H_z^{\,j+0.5} - H_z^{\,j-0.5} \right)$$

$$E_y \mathrel{-}= \frac{\Delta t}{\Delta x\, \varepsilon_r \varepsilon_0}\left(H_z^{\,i+0.5} - H_z^{\,i-0.5} \right)$$



2) at time $t+0.5$: Update H fields everywhere using spatial derivatives of E

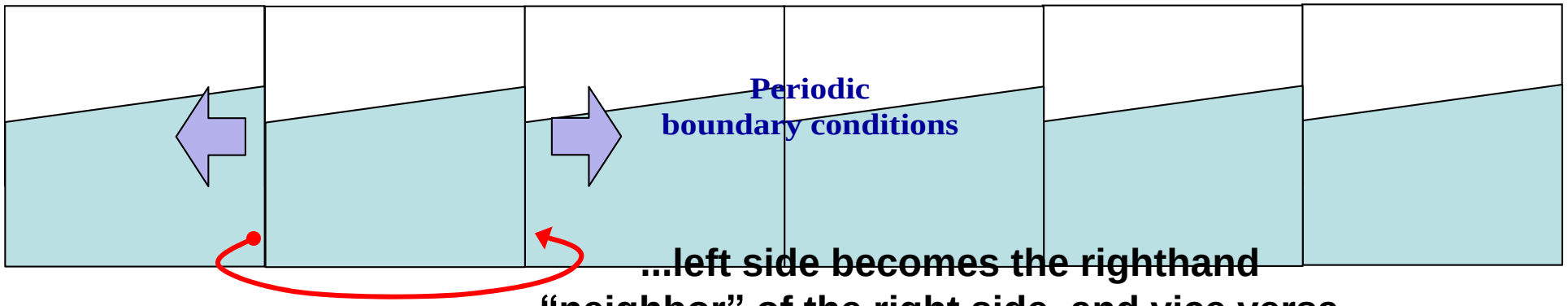
$$H_z \mathrel{+}= \frac{\Delta t}{\mu}\left(\frac{E_x^{\,j+1} - E_x^{\,j}}{\Delta y} + \frac{E_y^{\,i} - E_y^{\,i+1}}{\Delta x} \right)$$



- Every cell must get updated
- Accuracy requires $\Delta x, \Delta y, \Delta z \longrightarrow 0$
- Small $\Delta x, \Delta y, \Delta z$ forces small Δt (Courant stability)
- What happens at edges? $\longrightarrow$ Boundary conditions

Boundary conditions

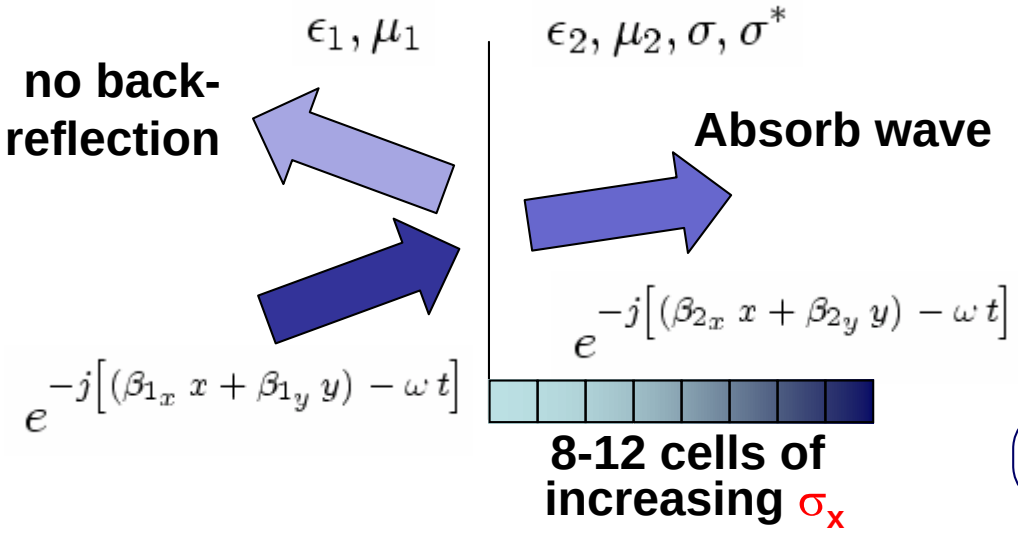
Infinite but periodic structures



...left side becomes the righthand
"neighbor" of the right side, and vice versa...

Isolated structures *(by absorbing outgoing radiation)*

"Perfectly Matched Layer" (PML)



Back-reflection $\propto \eta_1 \cos\theta - \eta_2 \cos\theta$ ($\sigma_y = 0$)

$$\eta_1 = \sqrt{\frac{\mu_1}{\epsilon_1}} \qquad \eta_2 = \sqrt{\frac{\mu_2}{\epsilon_2}\,\frac{(1+\sigma_x{}^*/j\omega\mu_2)}{(1+\sigma_x/j\omega\epsilon_2)}}$$

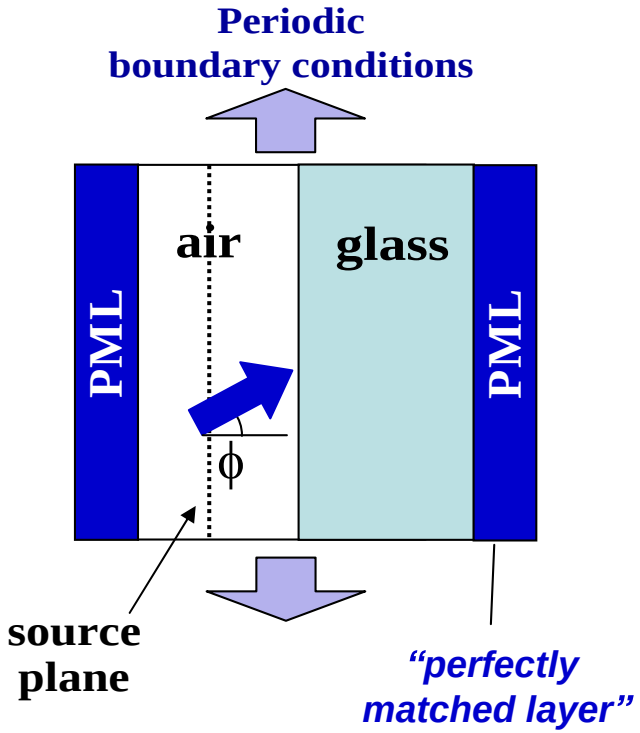
$$\epsilon_2 \frac{\partial E_x}{\partial t} + \sigma_y E_x = \frac{\partial H_z}{\partial y}$$

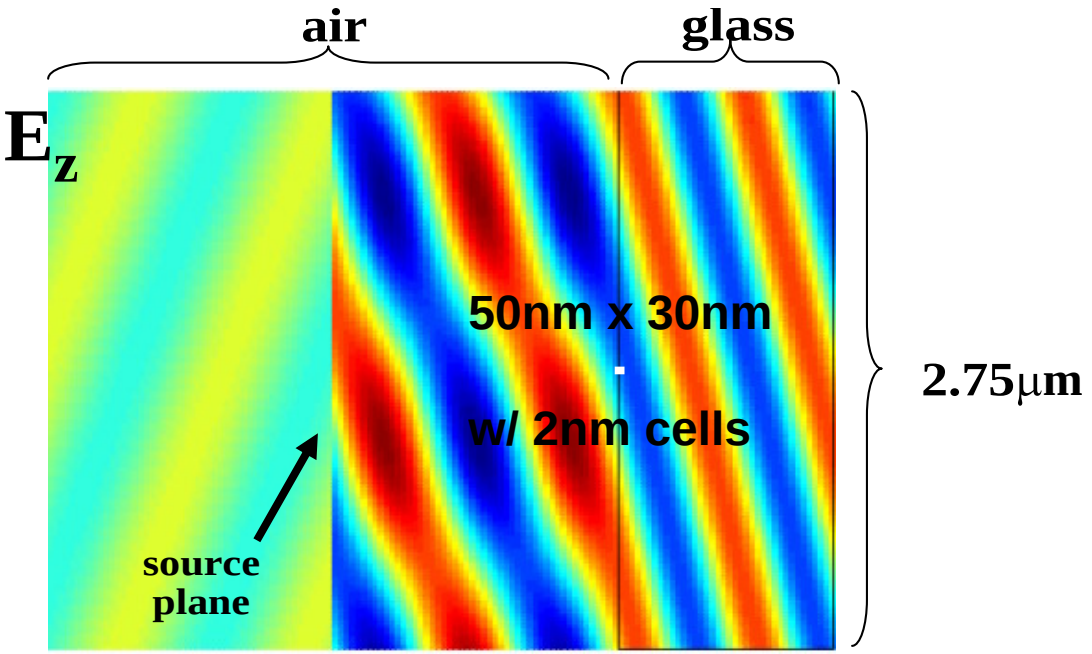
$$\epsilon_2 \frac{\partial E_y}{\partial t} + \sigma_x E_y = -\frac{\partial H_z}{\partial x}$$

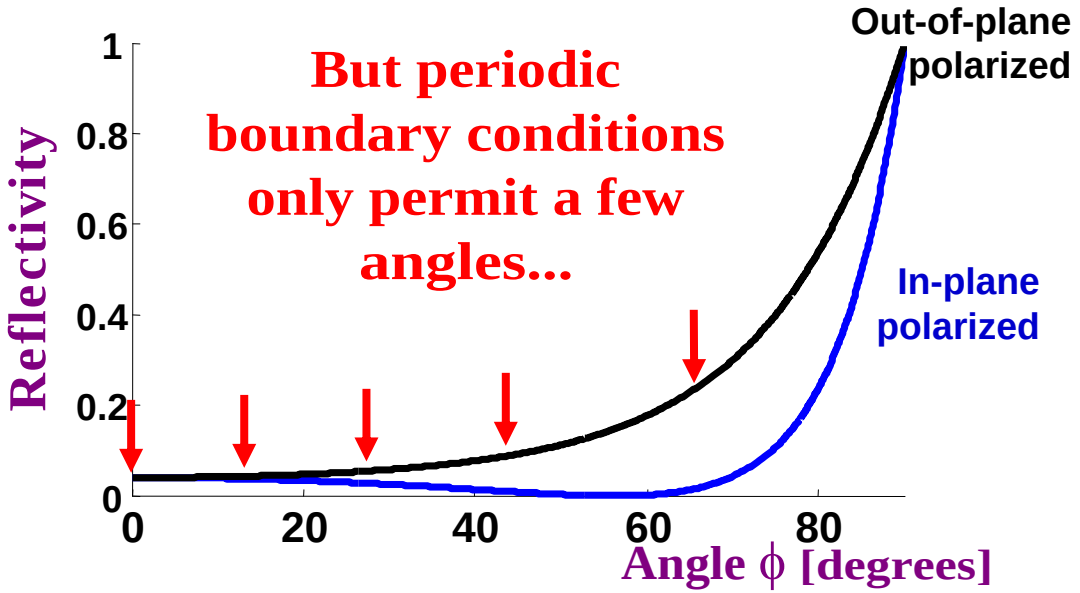
$$R(\theta) = \exp(-2\,\sigma_x\,\eta_2\,d\,\cos\theta)$$

Sample simulation (validation)

Fresnel reflection coefficients with FDTD



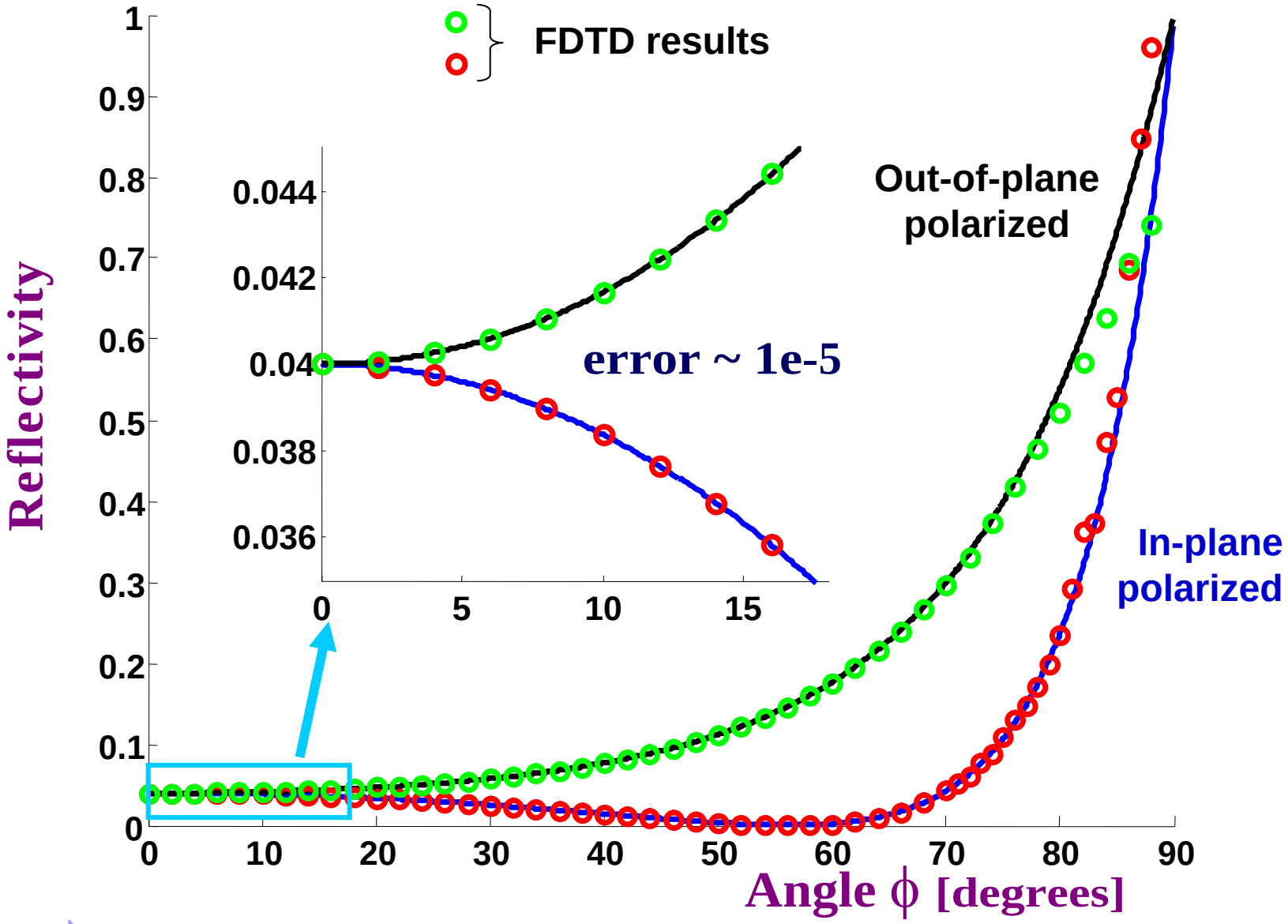




Solution: "Floquet" boundary conditions:

(build 2-part sim: 1@sin(ωt), 1@cos(ωt) – exchange at boundary as exp(±*j* ***k*** · ***L***)

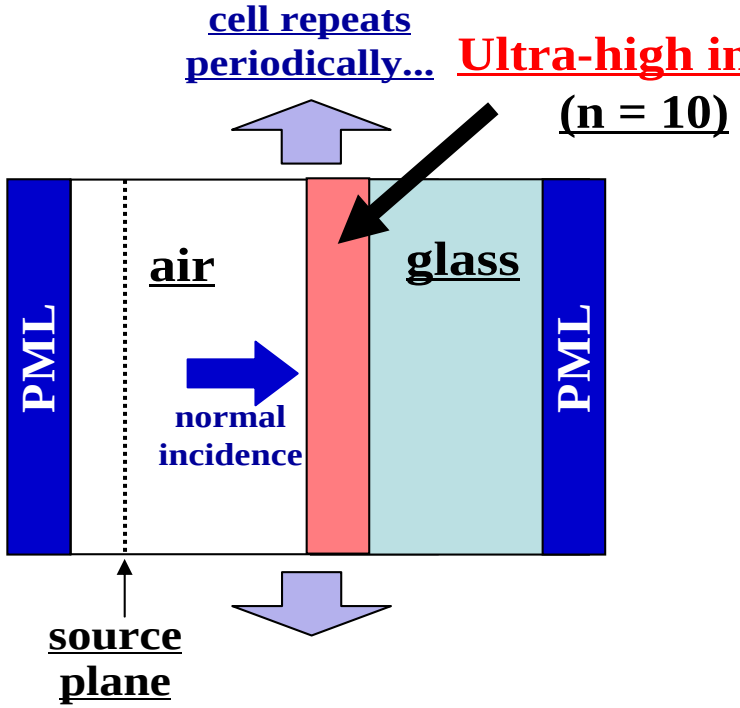
Fresnel reflection coefficients

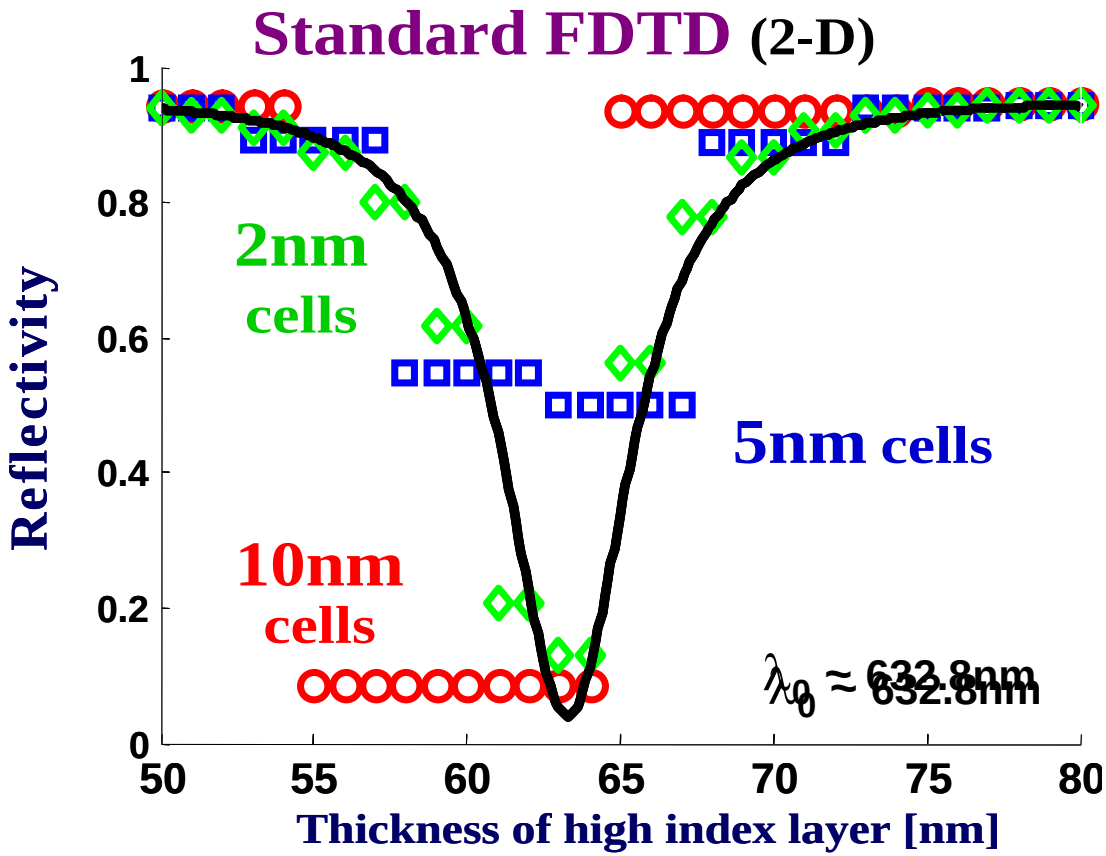


Error analysis: for 1% error, need cell-size < λ/50

What happens with coarser gridding?

An extreme example: a thin-film of unphysically high index





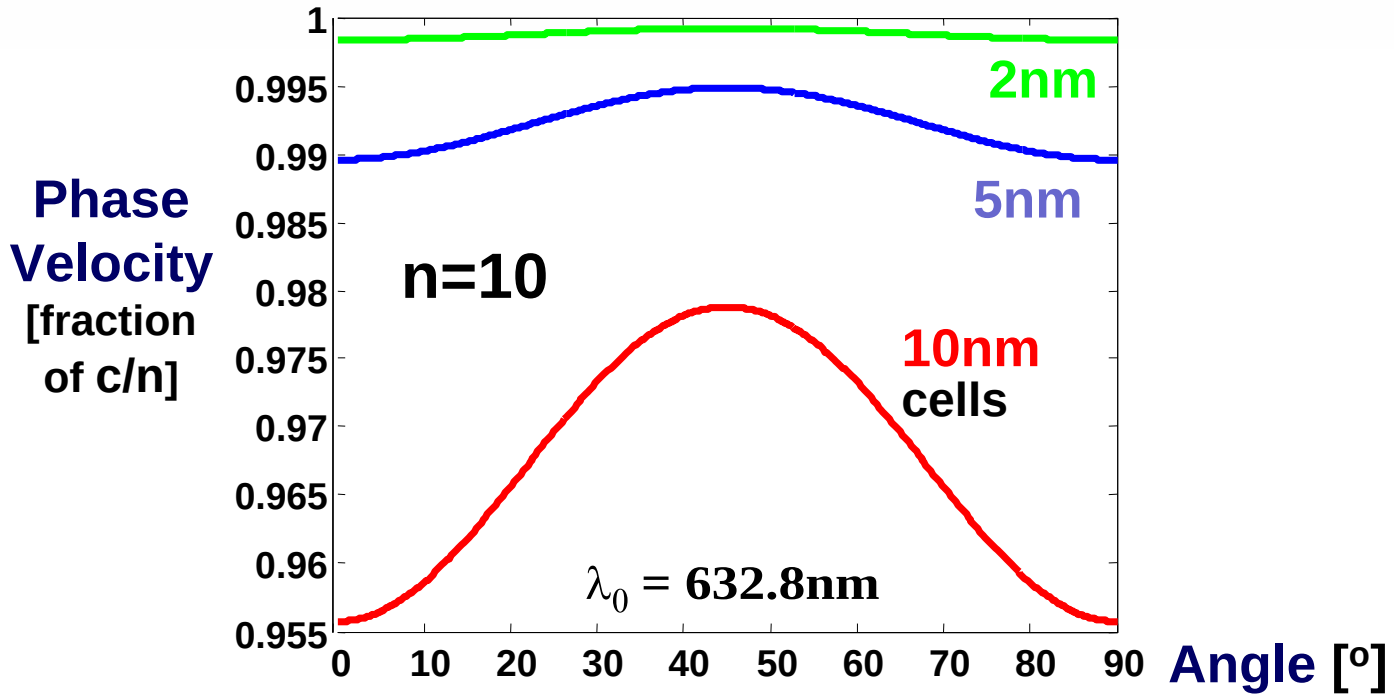
- small spatial features get lost
- get wrong answer AND wrong local minimum

"Numerical dispersion"

Dispersion relation should be: $\left(\frac{\omega}{c}\right)^2 = (k_x)^2 + (k_y)^2 + (k_z)^2$

but instead is...

$$\left[\frac{1}{c\Delta t}\sin\left(\frac{\omega\Delta t}{2}\right)\right]^2 = \left[\frac{1}{\Delta x}\sin\left(\frac{\tilde{k_x}\Delta x}{2}\right)\right]^2 + \left[\frac{1}{\Delta y}\sin\left(\frac{\tilde{k_y}\Delta y}{2}\right)\right]^2 + \left[\frac{1}{\Delta z}\sin\left(\frac{\tilde{k_z}\Delta z}{2}\right)\right]^2$$

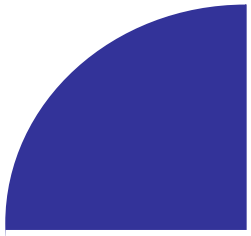


Partial solution: when index is high, adjust local μ_0 and ε_0 (increasing speed of light w/o affecting η)

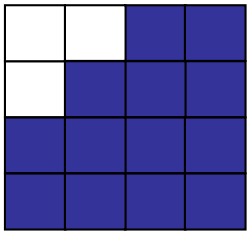
But only an exact fix for one angle & frequency...

"Staircasing" of small spatial features

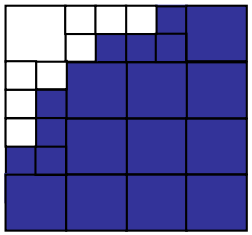
We really want to simulate a smooth interface between regions of ε_1 and ε_2...



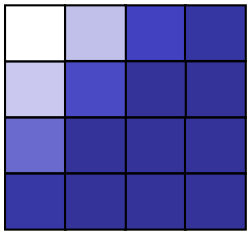
...but the discretization in FDTD gives a "staircase" effect.



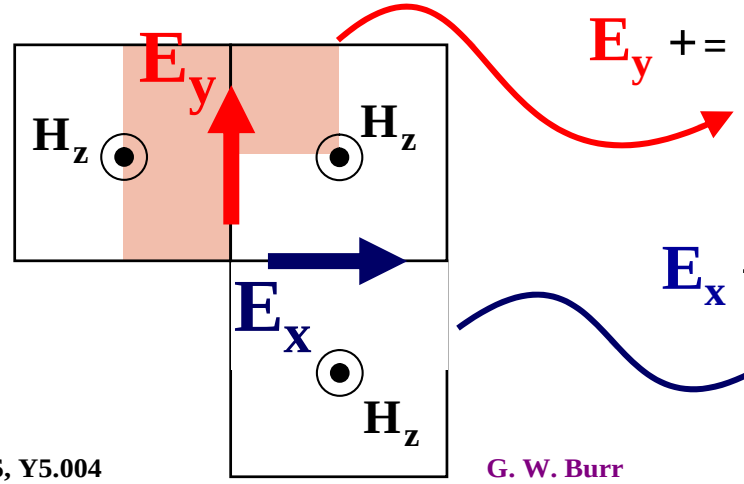
Using small grid cells only where we need them sounds ideal, but the logistics are unpleasant...



...but we can get a similar effect by using an *effective* ε in each cell.



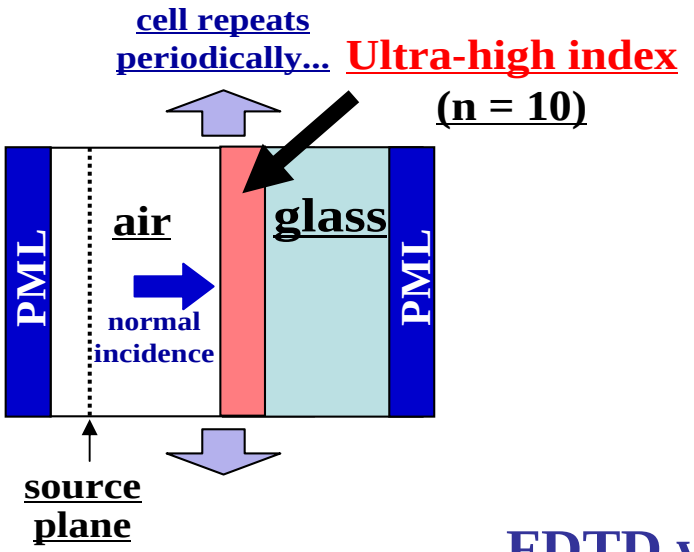
Each E-field component gets its own effective ε, which makes the most sense in terms of the field-update equations.

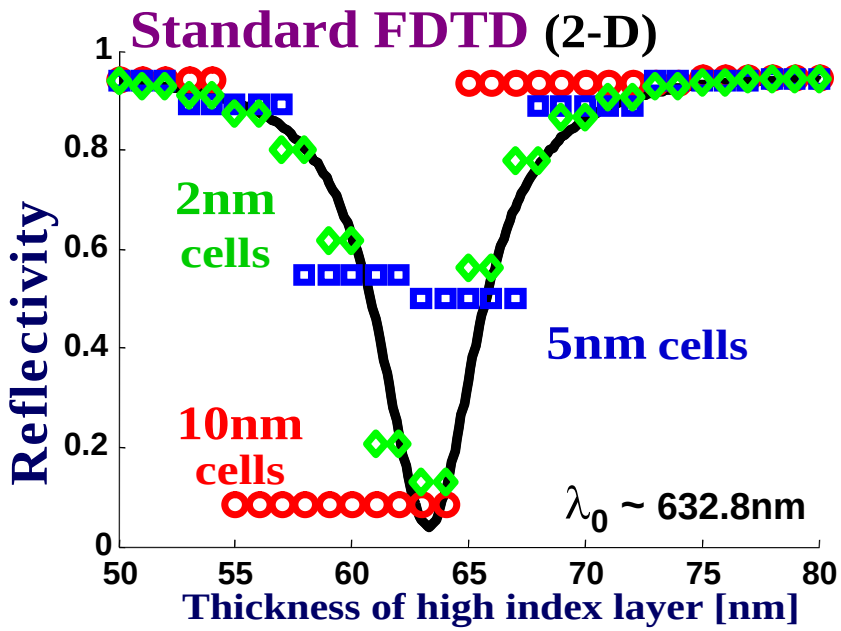


$$E_y \mathrel{+}= \frac{\Delta t}{\varepsilon_r \varepsilon_0} \left(\frac{H_z - H_z^{\text{previous cell}}}{\Delta x} \right)$$

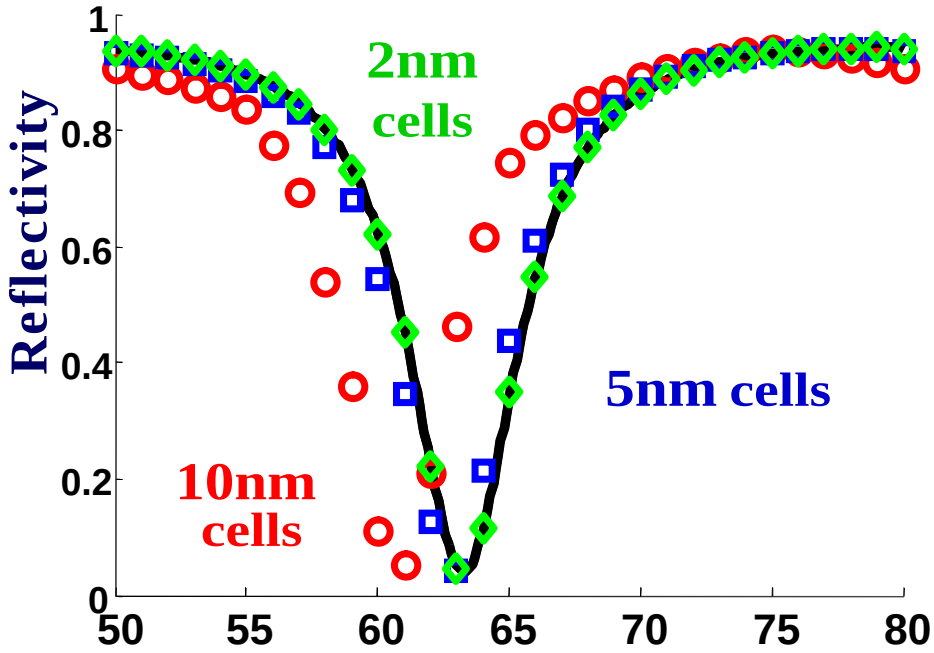
$$E_x \mathrel{+}= \frac{\Delta t}{\varepsilon_r \varepsilon_0} \left(\frac{H_z - H_z^{\text{previous cell}}}{\Delta y} \right)$$

Returning to our artificial example...





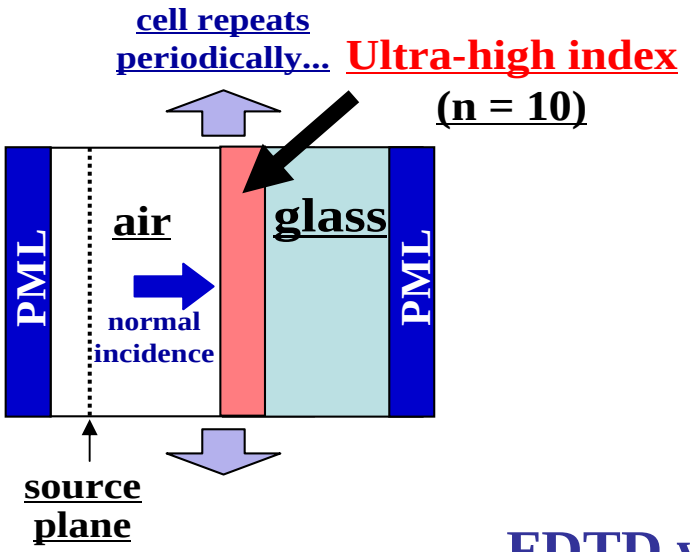
FDTD with field-based effective ε

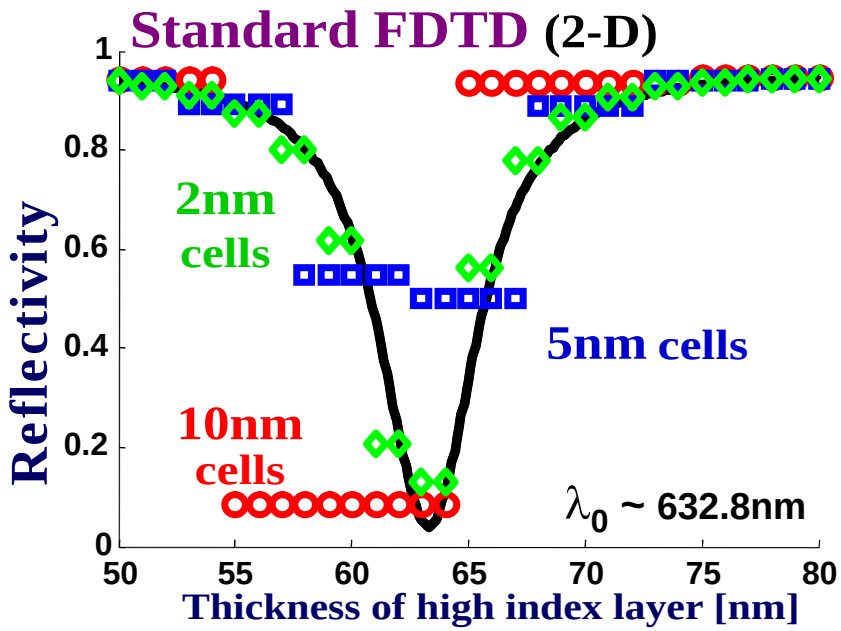


Despite coarse gridding...

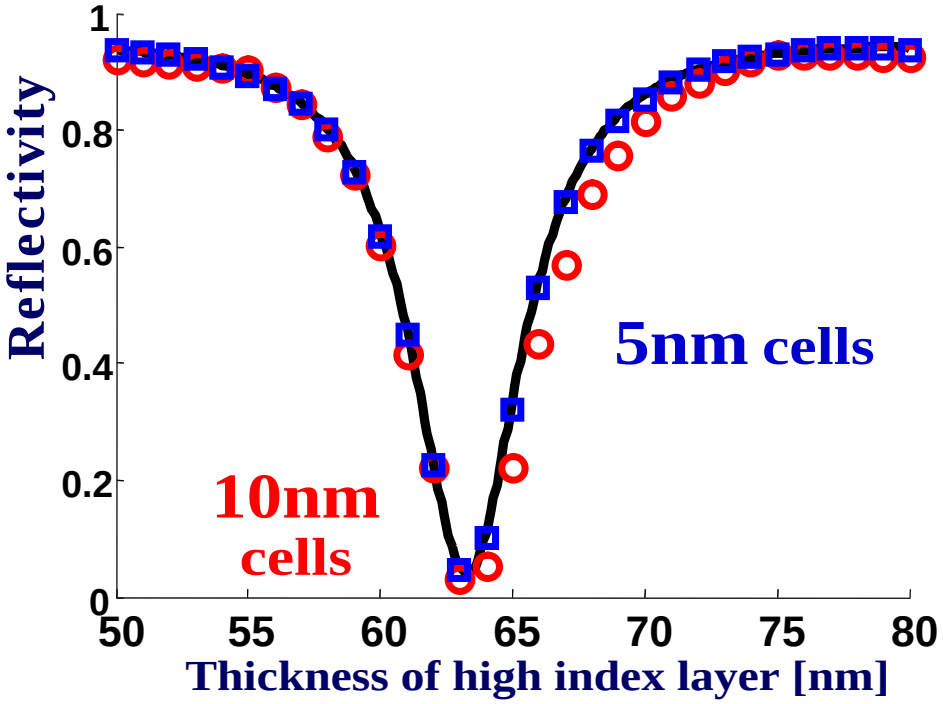
- small spatial features get modeled
- get the correct answer AND the right local minimum

Returning to our artificial example...





FDTD with field-based effective ε *and* (ε_0, μ_0) corrected for numerical dispersion



Despite coarse gridding...

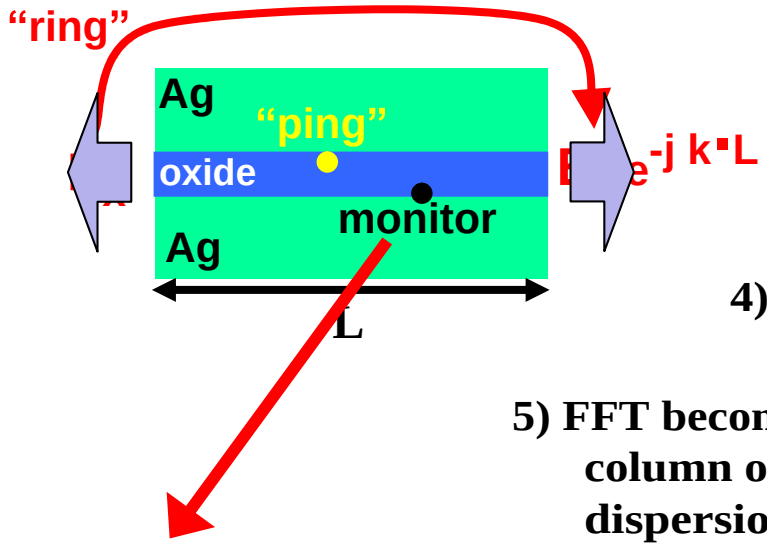
- small spatial features get modeled
- get the correct answer AND the right local minimum

Measuring frequency response

Example: Mapping out a dispersion diagram...

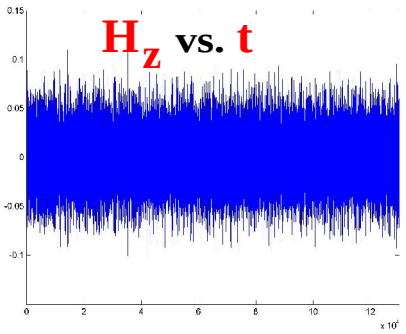
1) Define unit cell

2) Ping it with an impulse

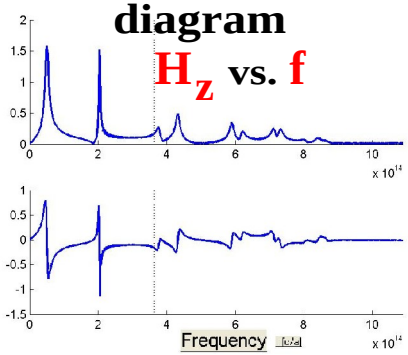


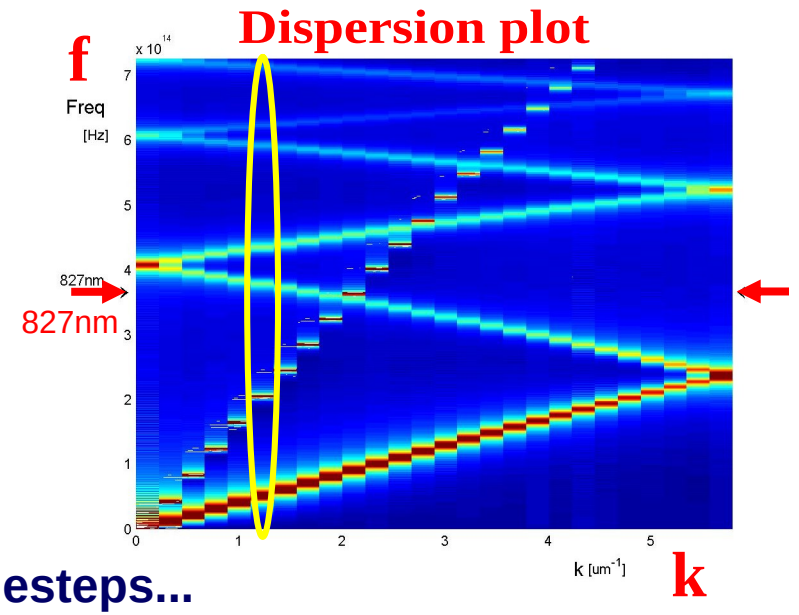
3) Set k-vector at boundaries

4) monitor at non-symmetry point



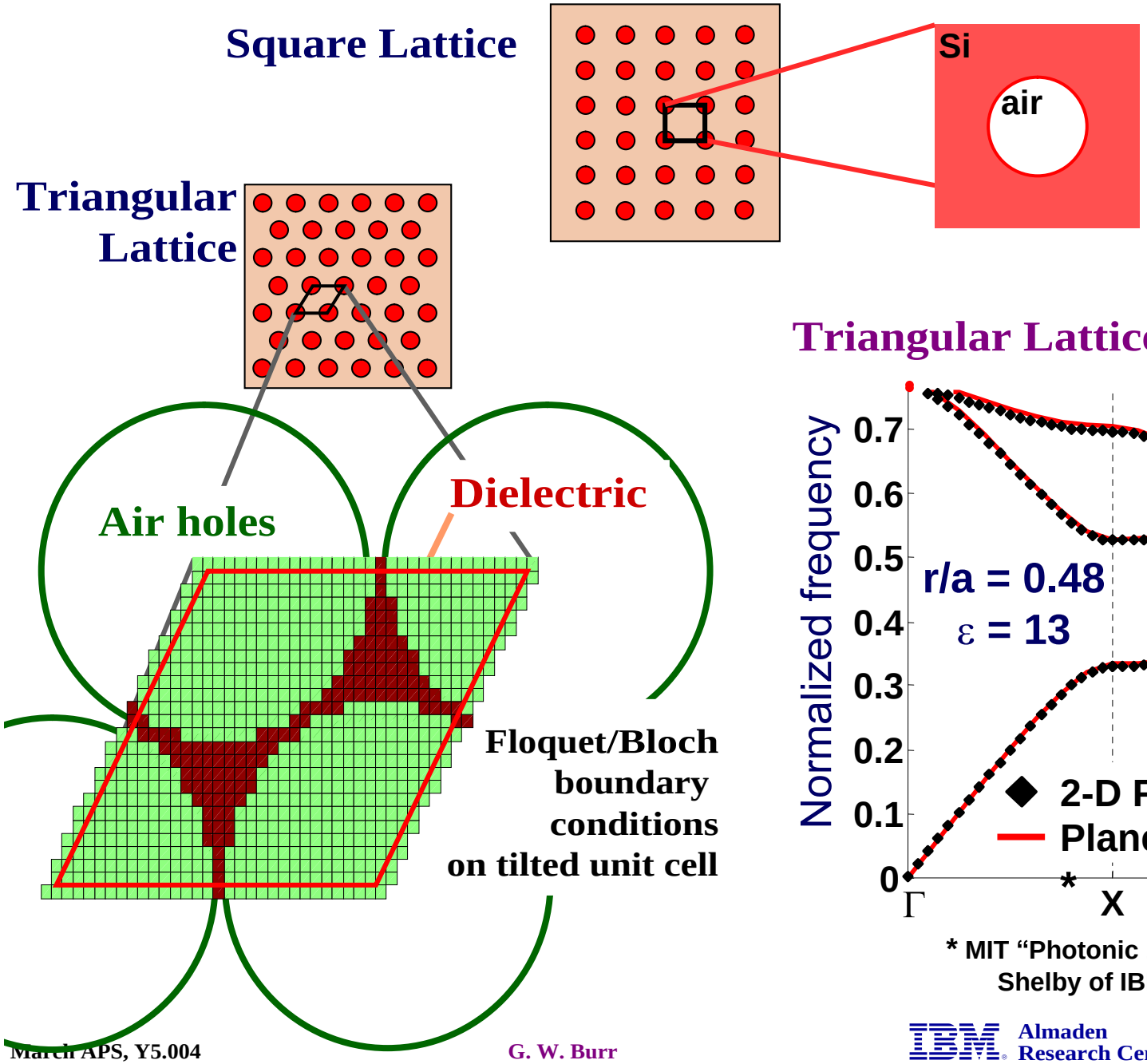
5) FFT becomes a column of the dispersion diagram



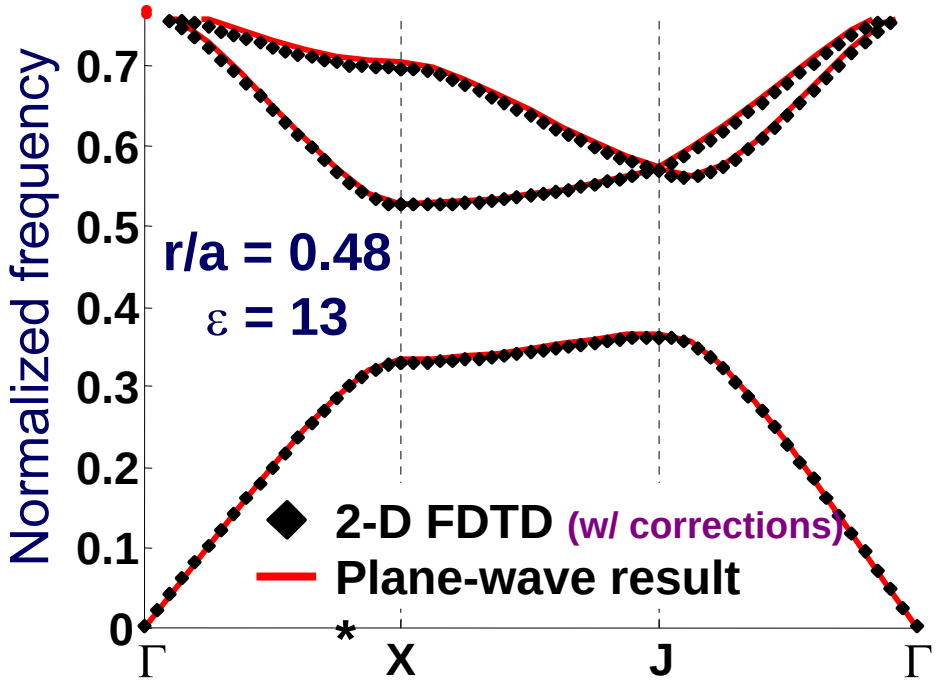


- For good frequency resolution, need lots of timesteps...

Example: photonic crystal band-diagrams



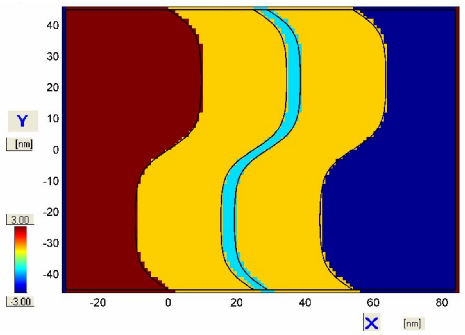
Triangular Lattice – TE bands



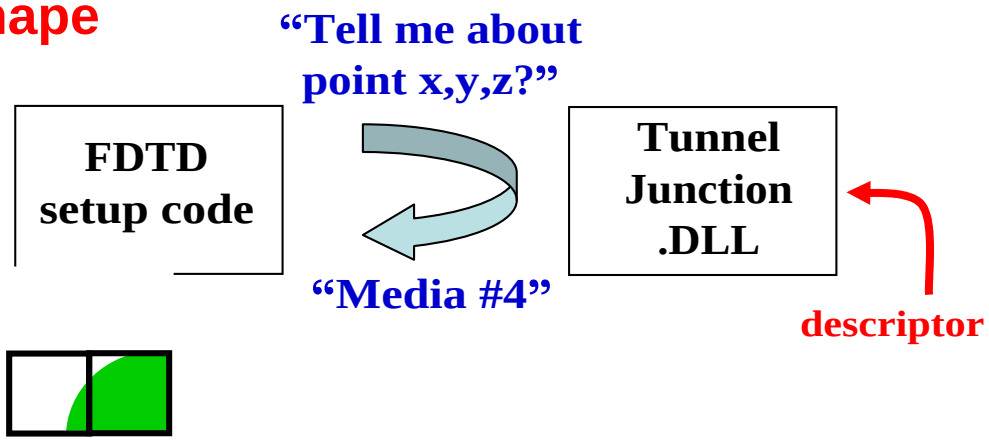
* MIT "Photonic bands" code – thanks to Bob Shelby of IBM Almaden for his help...

Custom functionality by plug-in DLL

Arbitrary structures



ArbShape

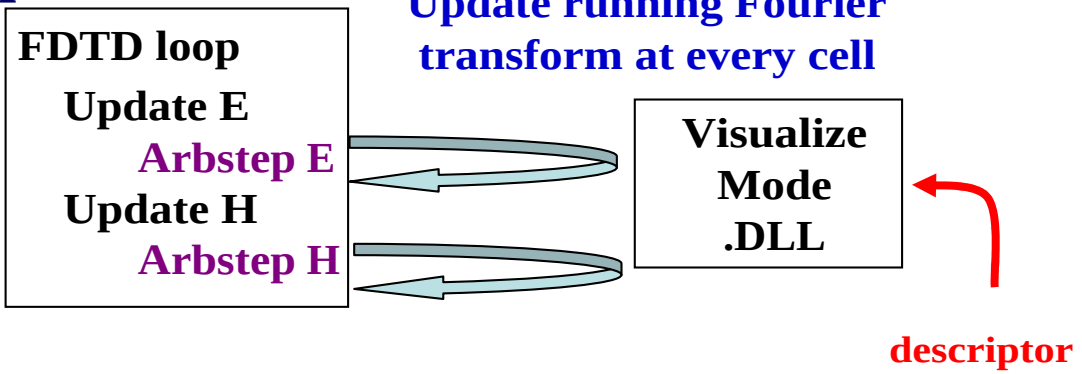


same code can gauge effective index...

Arbitrary computational steps

- VisualizeMode
- TriangularLattice

ArbStep

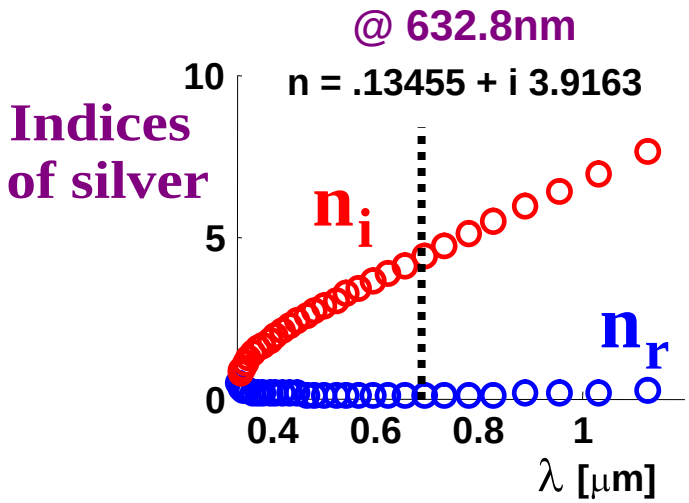


Metals

FDTD cannot model materials with Re{ε} < 1

(such as silver at visible wavelengths)

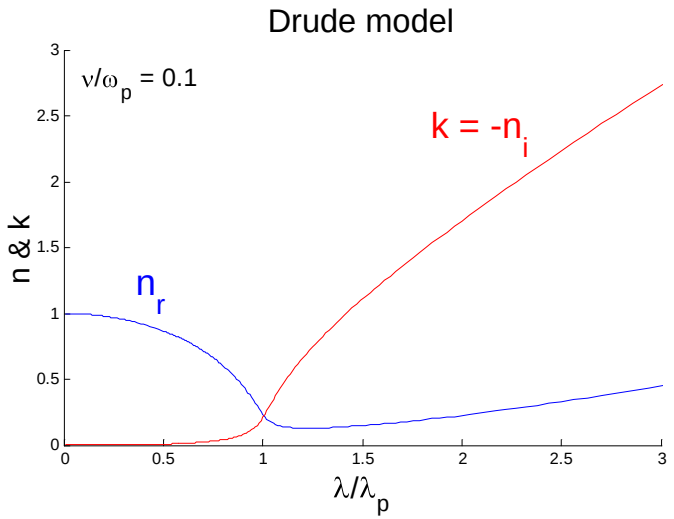
except by also modeling material dispersion.



$$\epsilon_\infty \frac{\partial \vec{E}}{\partial t} = \nabla \times \vec{H} - \vec{J}_p$$

$$\frac{\partial \vec{P}}{\partial t} = \vec{J}_p$$

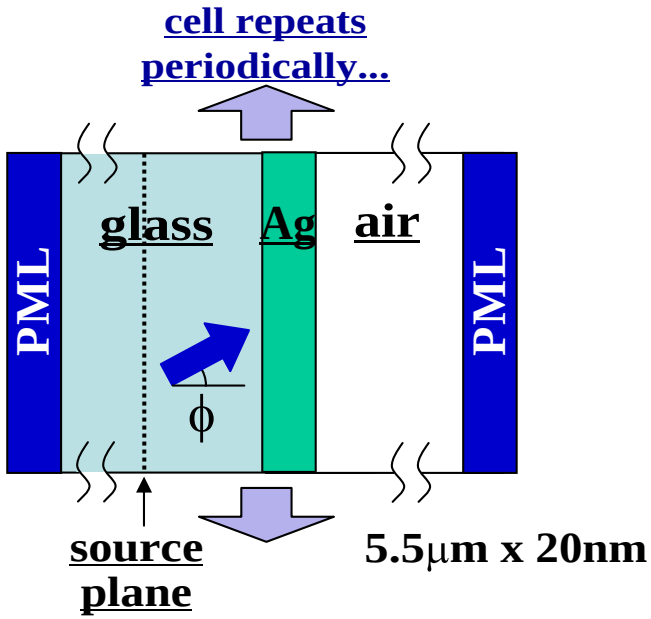
$$\frac{\partial \vec{J}_p}{\partial t} + \vec{J}_p = f\left(\vec{E}, \vec{P}, |E|^2, \text{etc.}\right)$$

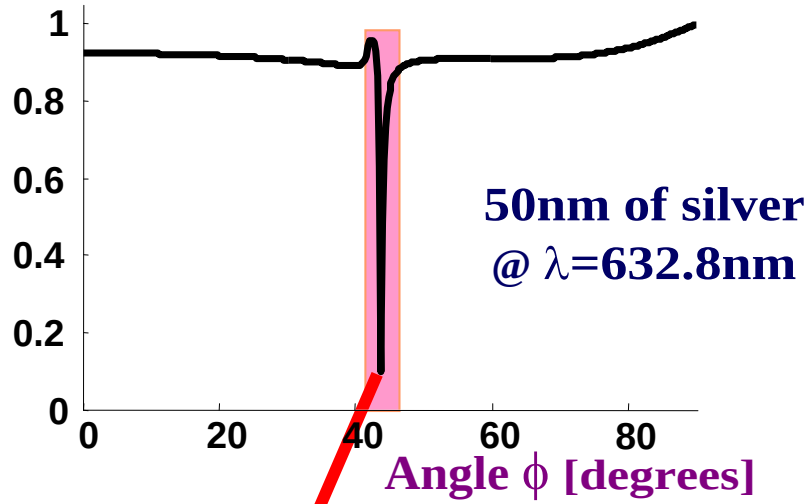


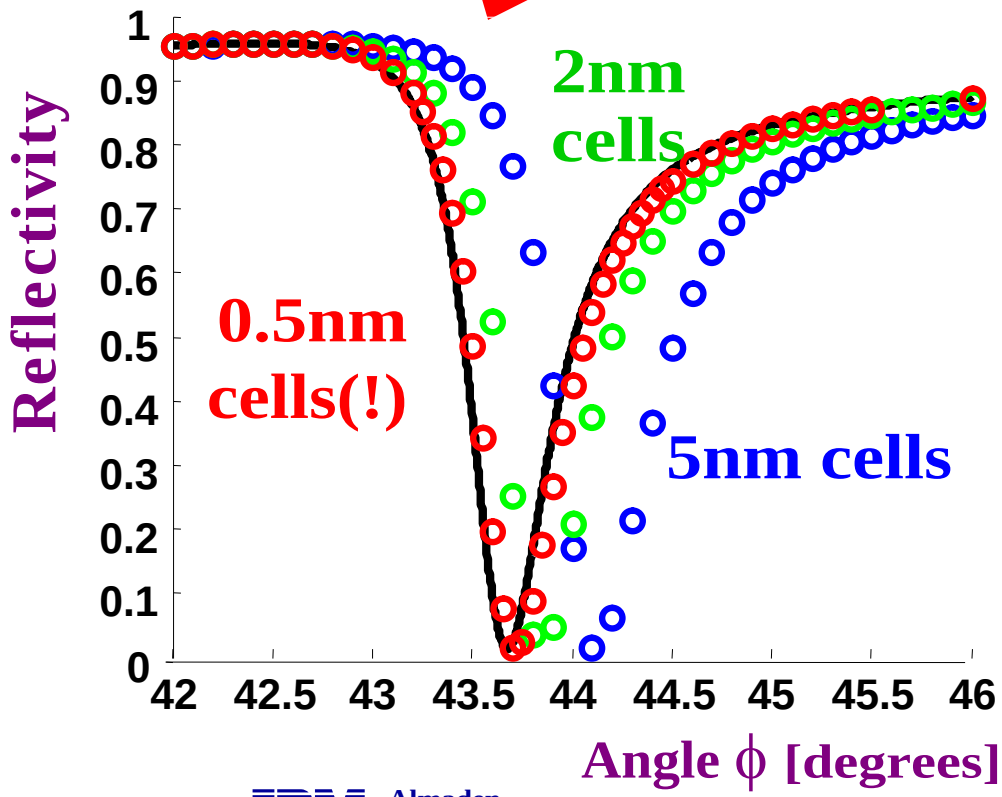
- **extra variables & update rules at each metal/dispersive cell**

Metals - verification

Plasmon resonance is extremely sharp:







FDTD in summary

Advantages:

- **algorithm is rigorous — potential for arbitrarily high accuracy**
- **can handle dispersive material including metals (surface plasmons, etc.)**
- **time-domain simulation: one simulation can model broad frequency response**
- **simple core algorithm +**
  nearest-grid-neighbor dependencies — amenable to parallelization
- **FDTD can support**
  - **finite or infinitely-periodic structures**
  - **arbitrary spatial arrangements of materials**
  - **input of pulsed, CW, or impulse waveforms**
  - **point-source, plane-wave, or mode-profile wavefronts**
  - **measurement of field, intensity, Poynting vectors**

FDTD in summary

Disadvantages: *("no pain, no gain")*

- **FDTD only becomes accurate as cell-spacing → zero**
- **small cell spacing also mandates short time-steps (Courant stability)**
- **finite # of cells — boundary conditions at edges of the simulation are critical**
- **high frequency resolution requires many timesteps**
  (Fourier-transform relationship)
- **all simulation cells must be updated each timestep**
  — computing just a portion is inefficient

The sum effect:

**"to be confident you will get the right answer,
3-D FDTD simulations must be large & slow"**

But this means FDTD can only be a *design verification* tool,
never a *design optimization* tool

How to change this?

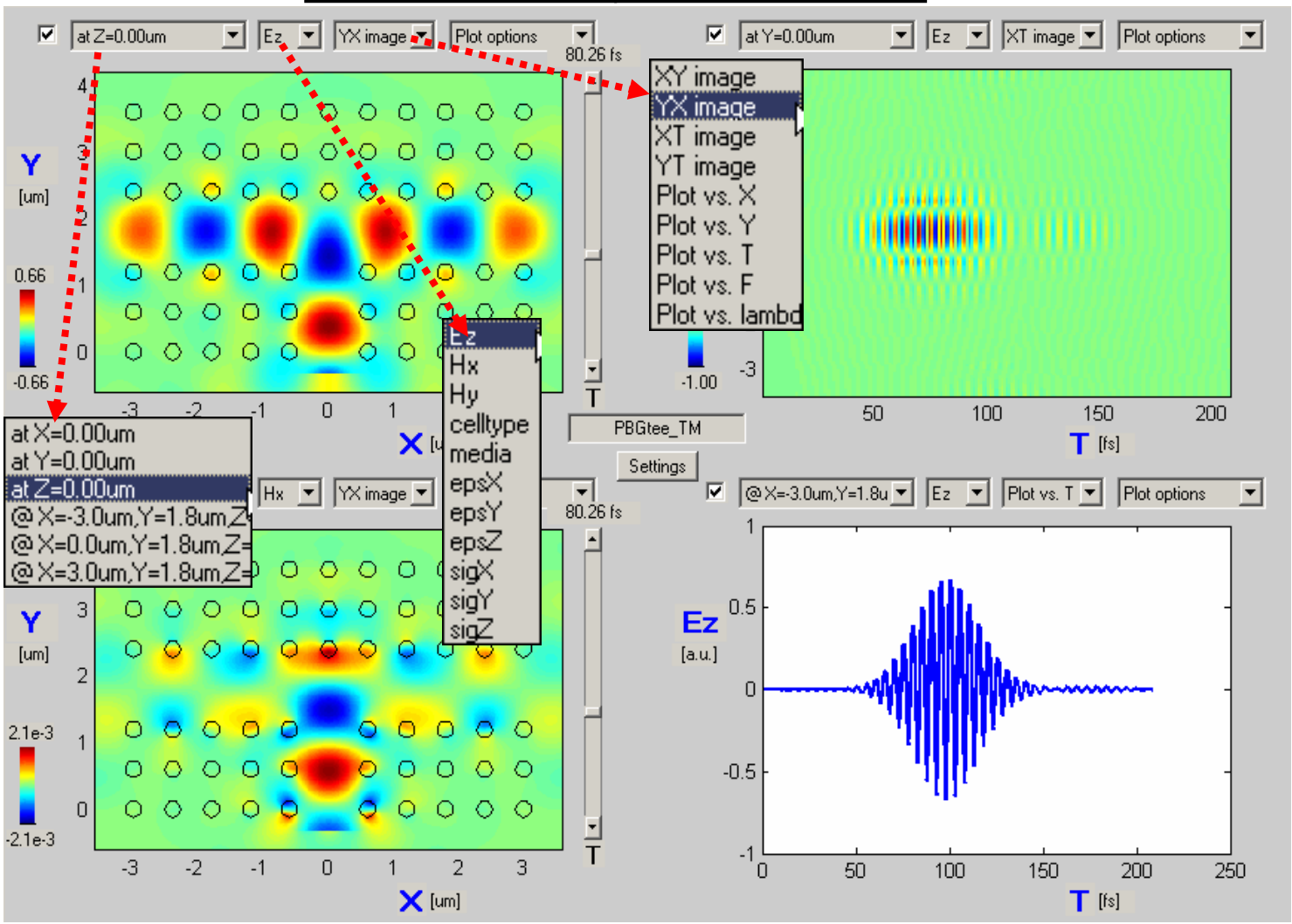
- **error mitigation**
- **careful design of numerical experiment**
- **grid computing!**

Matlab script

Parallelized C code

Single-processor C code

Matlab GUI



Notes for FDTD grid-implementation...

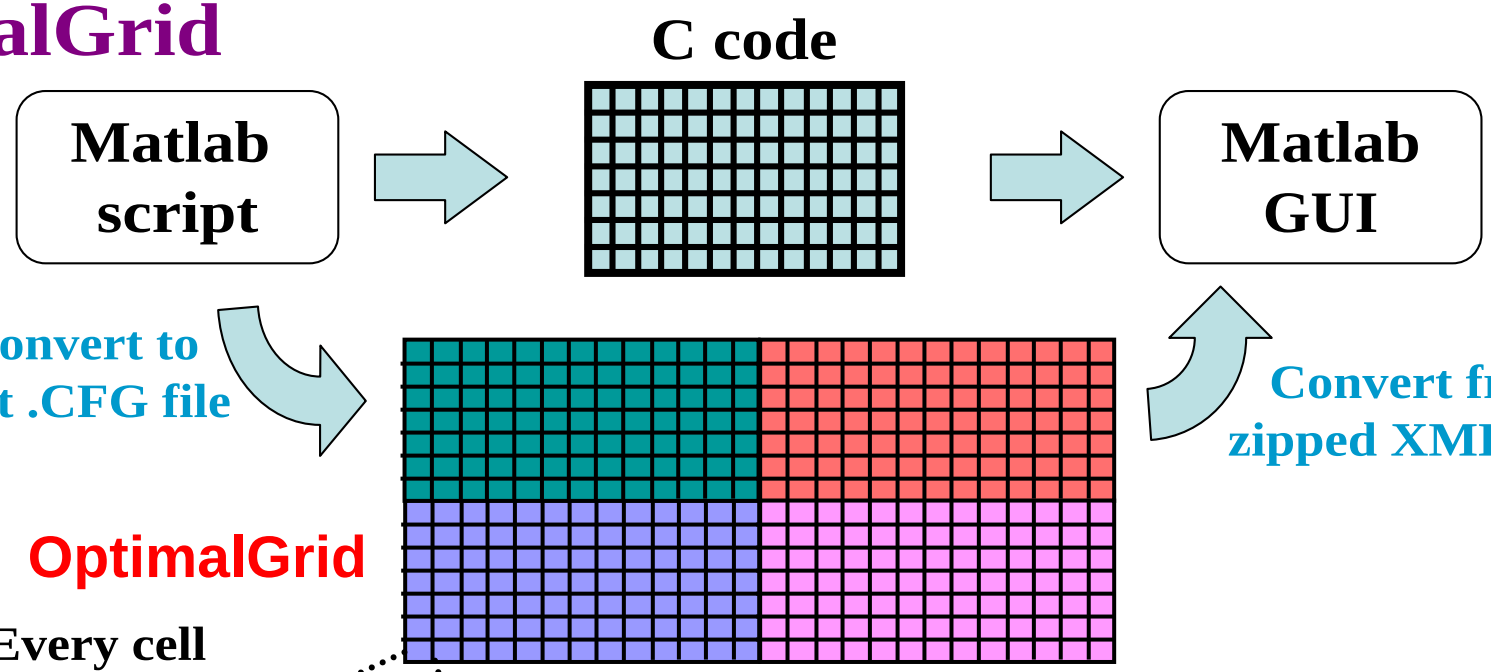
- want to keep same Matlab script for input definition,
  same Matlab GUI for output analysis

- some cells (PMLs, metals) may have more data payload than others
- may need to have opposite edges act like neighbors
- sometimes need 1 'extra' dimension (Floquet boundary conditions)
- support Plug-in DLLs

Also good to have:

- a GUI at input to *check* the simulation before running
- a quick way to estimate the
  memory usage, run-time, & file-space needs of a simulation
- minor feedback while a simulation is running...
- PCanywhere or equivalent, for remote administration

OptimalGrid



Pro:

- Simpler to program
- should scale well
- may support variable grid spacing

Con:

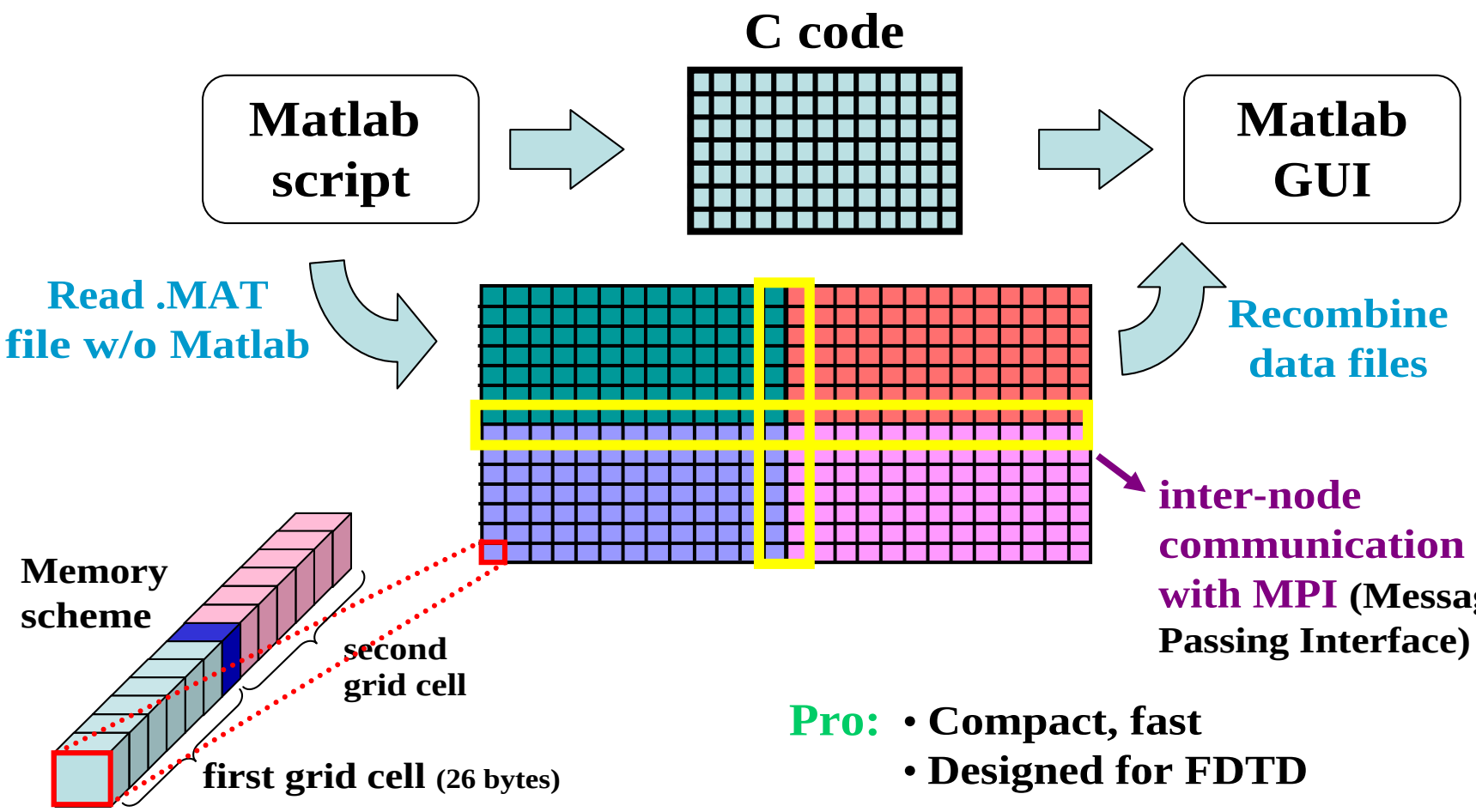
- Space & time overheads
- Java is slow

next step here is to move to JNI
(Java Native Interface)

James Kaufman
Toby Lehman
Glenn Deen
John Thomas
Zhenghua Fu

www.almaden.ibm.com/cs/OptimalGrid.html

Grid computing using MPI *(message-passing interface)*



- Each XY point is a separate pointer
- Data for Z spatial dimension (at that *x*,*y*) is all in one array
- Additional cell-data in separate "list" structure

Pro:
- Compact, fast
- Designed for FDTD

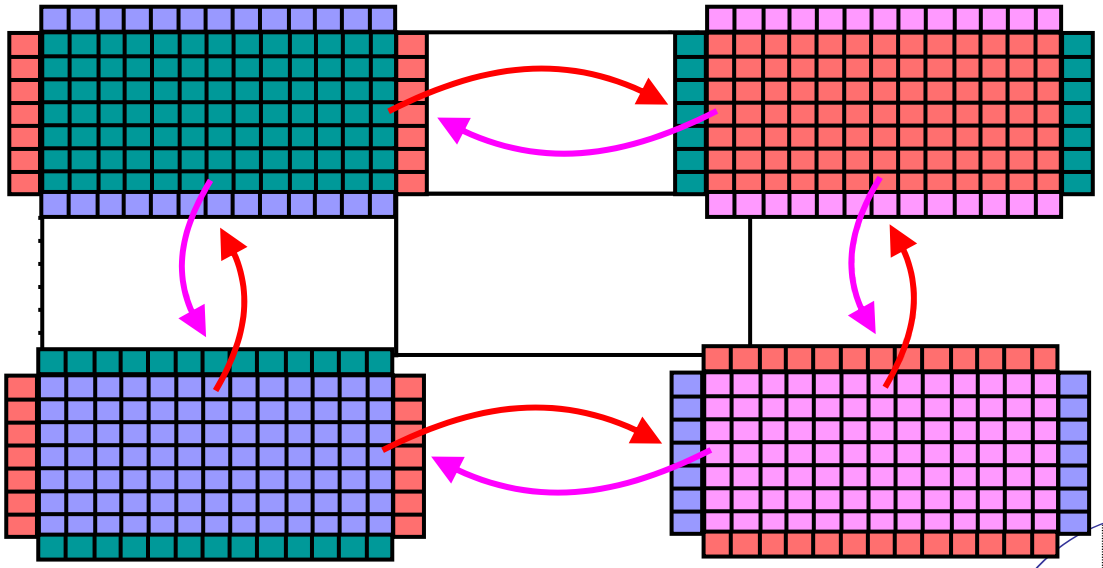
Con:
- Only 2-D mapping to computers
- Complexity (XY, XZ, YZ)
- Scalability?
- MPI implementation correct?
- How to have variable cell spacing?

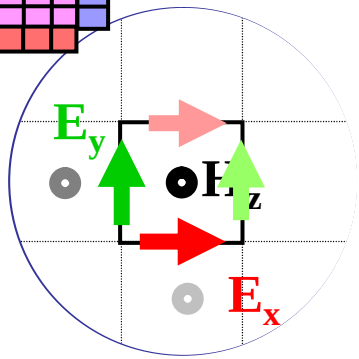
Grid computing using MPI

While each computer is responsible for one part of the problem-space...

every computer has the full 2-D array of pointers

Only the relevant pointers are allocated (rest are NULL)





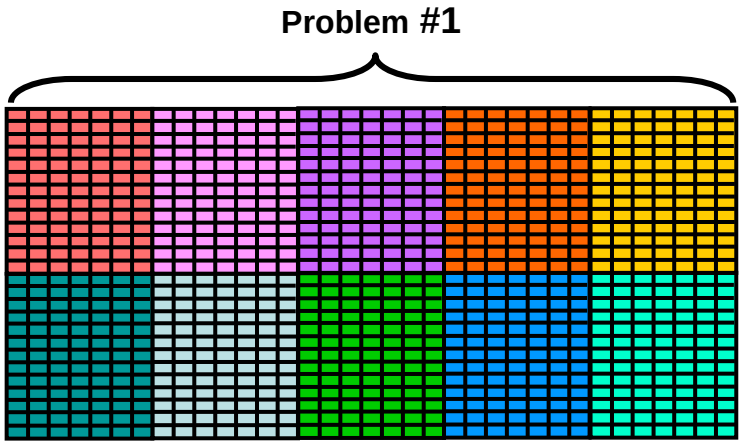
All redundant data is updated once per timestep, but in two portions – for instance, to update E-fields we only need info from "behind" for H-fields we need the info from "ahead"

To reduce the # of data transfers, relevant memory arrays sit in contiguous memory (where possible)

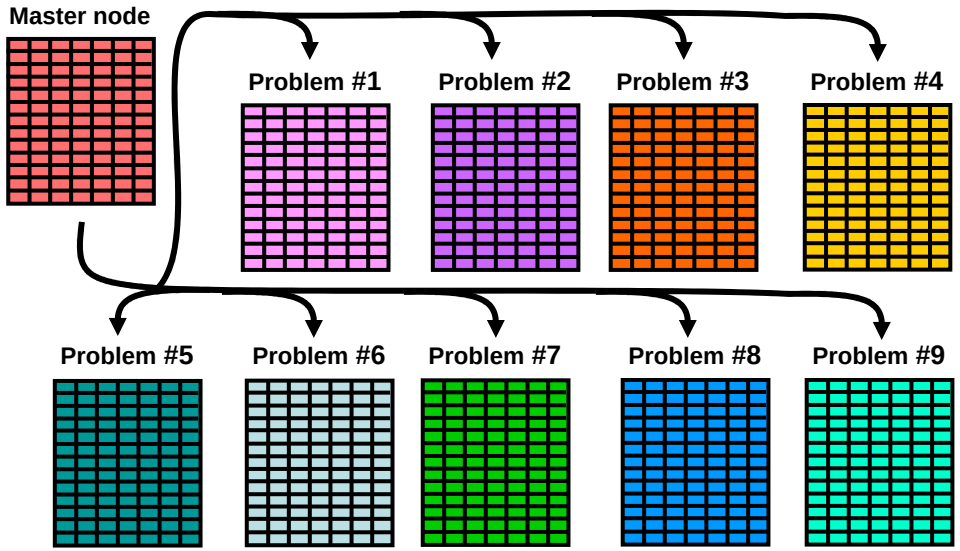
Plus periodic & Floquet boundary conditions!

Two ways to use this parallelism

Distributed mode: 1 job split amongst 10 machines

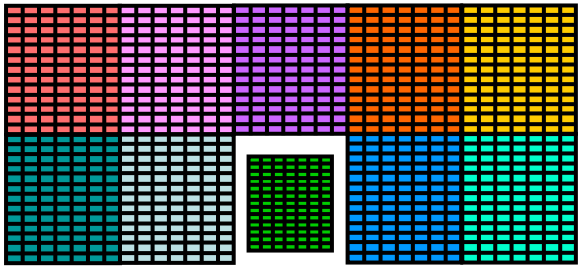


Master-slave mode: 9 jobs in parallel

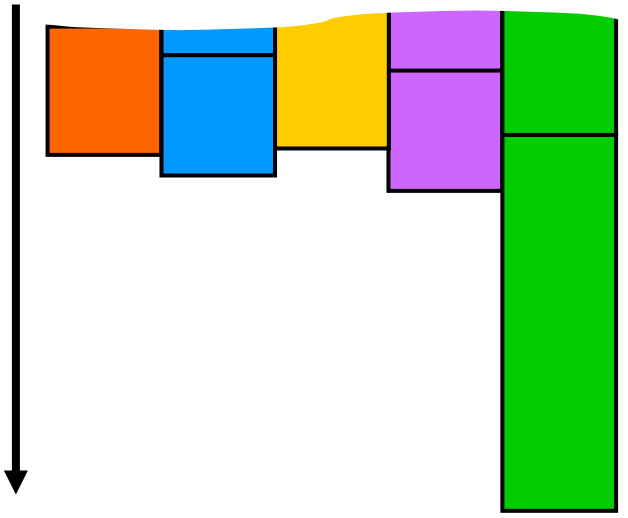


Problems with heterogeneous clusters:

Scaling to the worst performer:

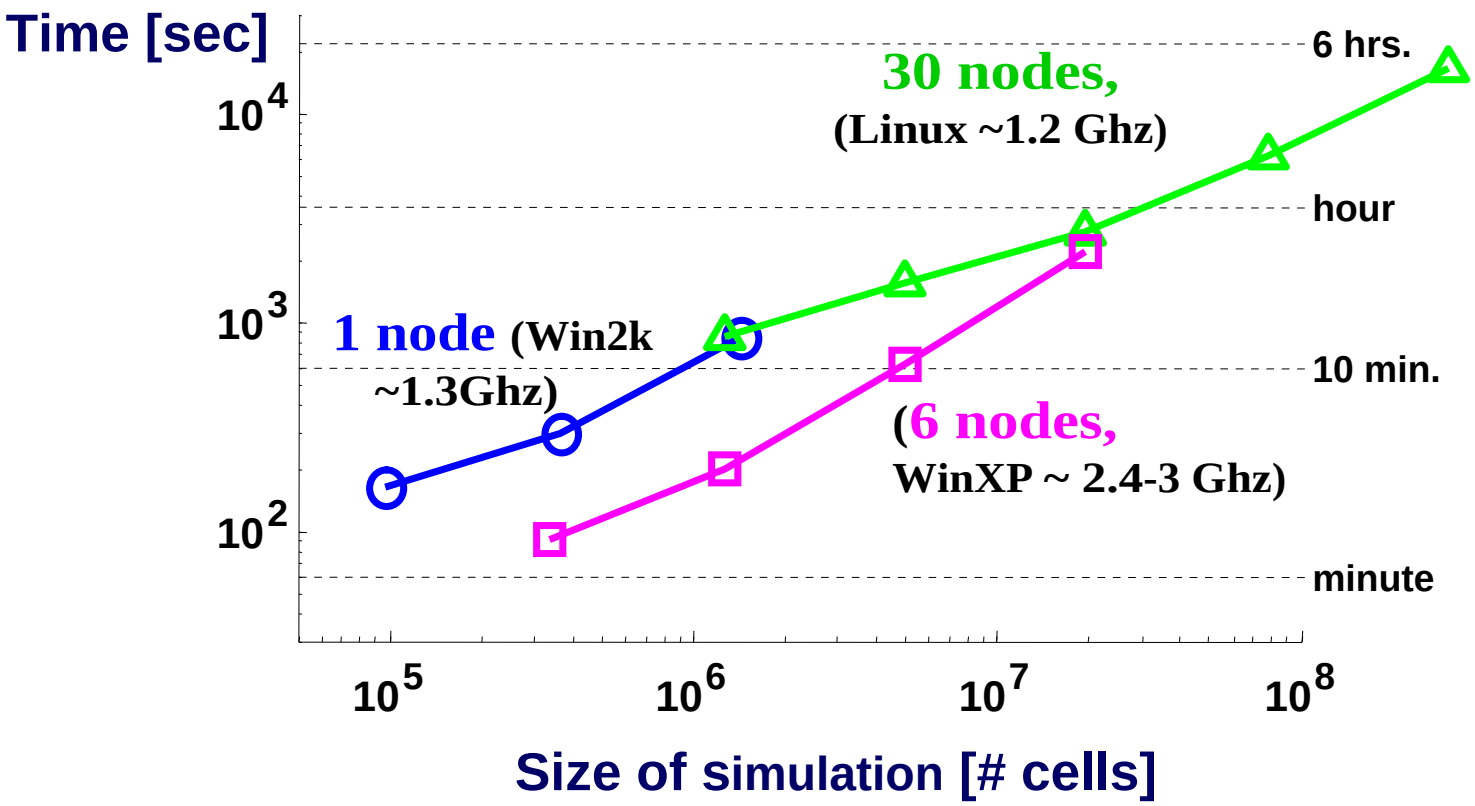


The curse of the last job:



Distributed mode...

...can handle really big simulations...

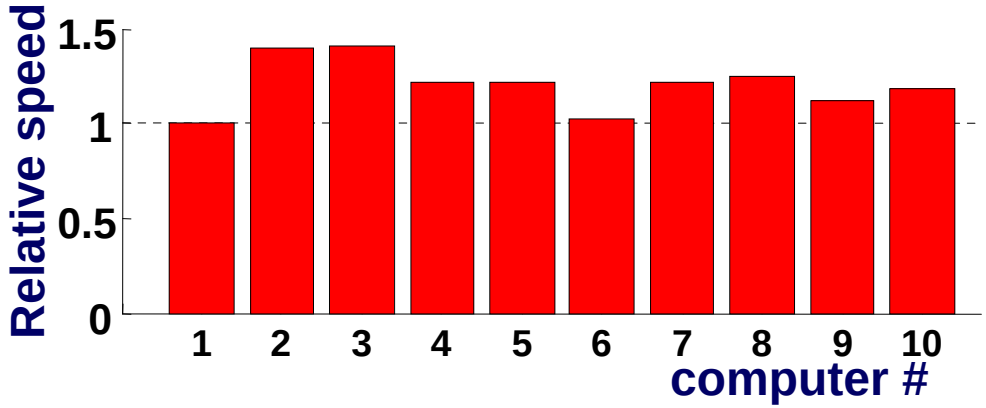


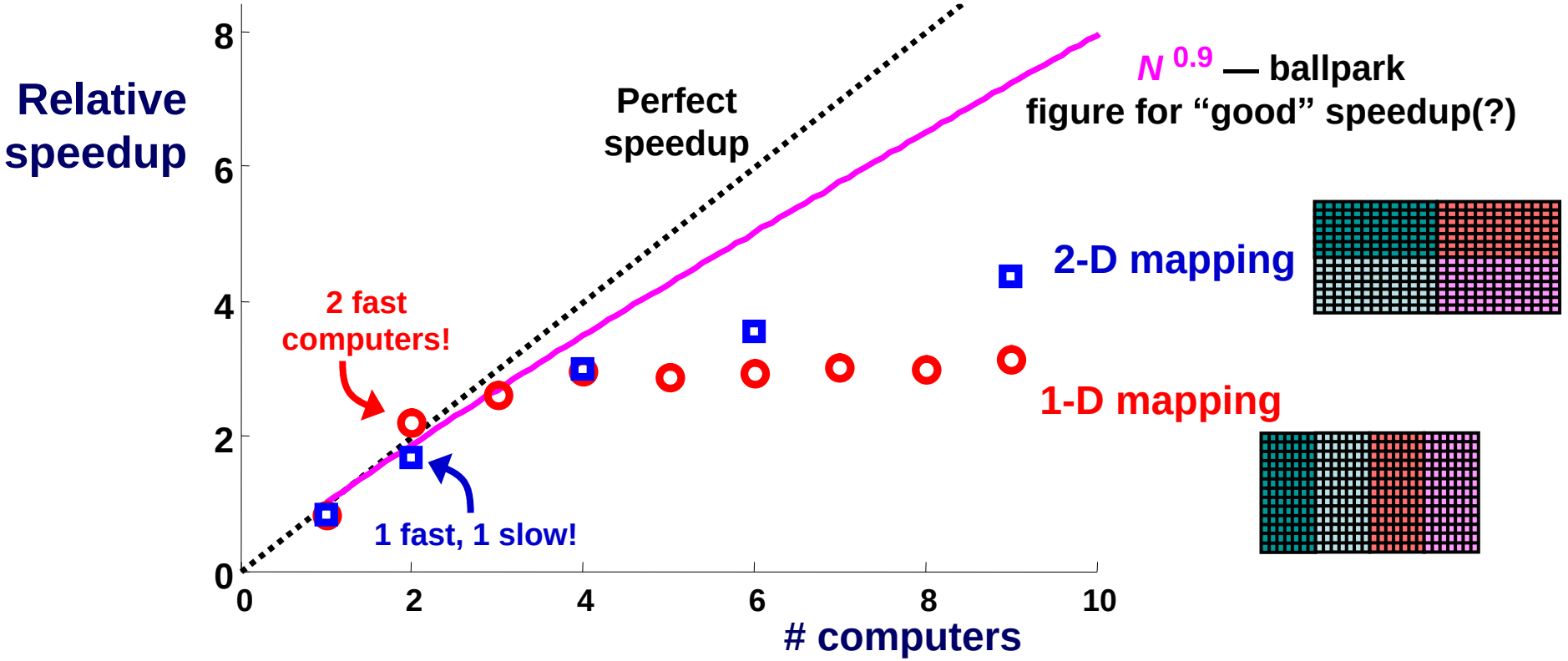
But how does the speedup scale?

Speed-up due to grid-computing

Heterogeneous grid:
Ten WinXP PCs,
2.4 – 3.1Ghz

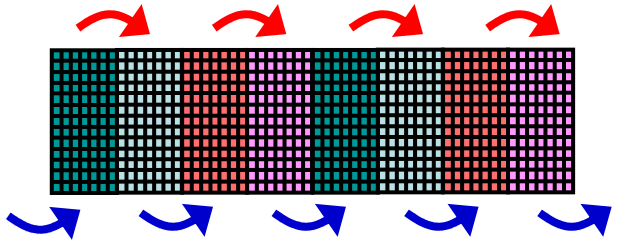
Gigabit ethernet





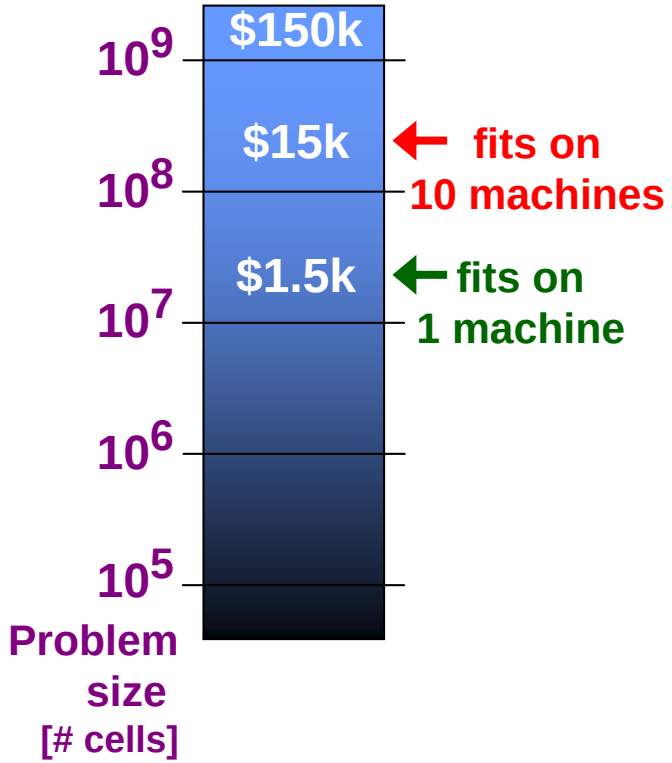
So where's the missing speedup?

- Need 3-D mapping to computers?
- Coding problems?
  - portioning of cells between slow & fast computers?
  - ordering of data exchanges?
  - incorrect use of synchronous/asynchronous MPI features?
- Shared-memory multi-processor system



I prefer Master-slave mode if

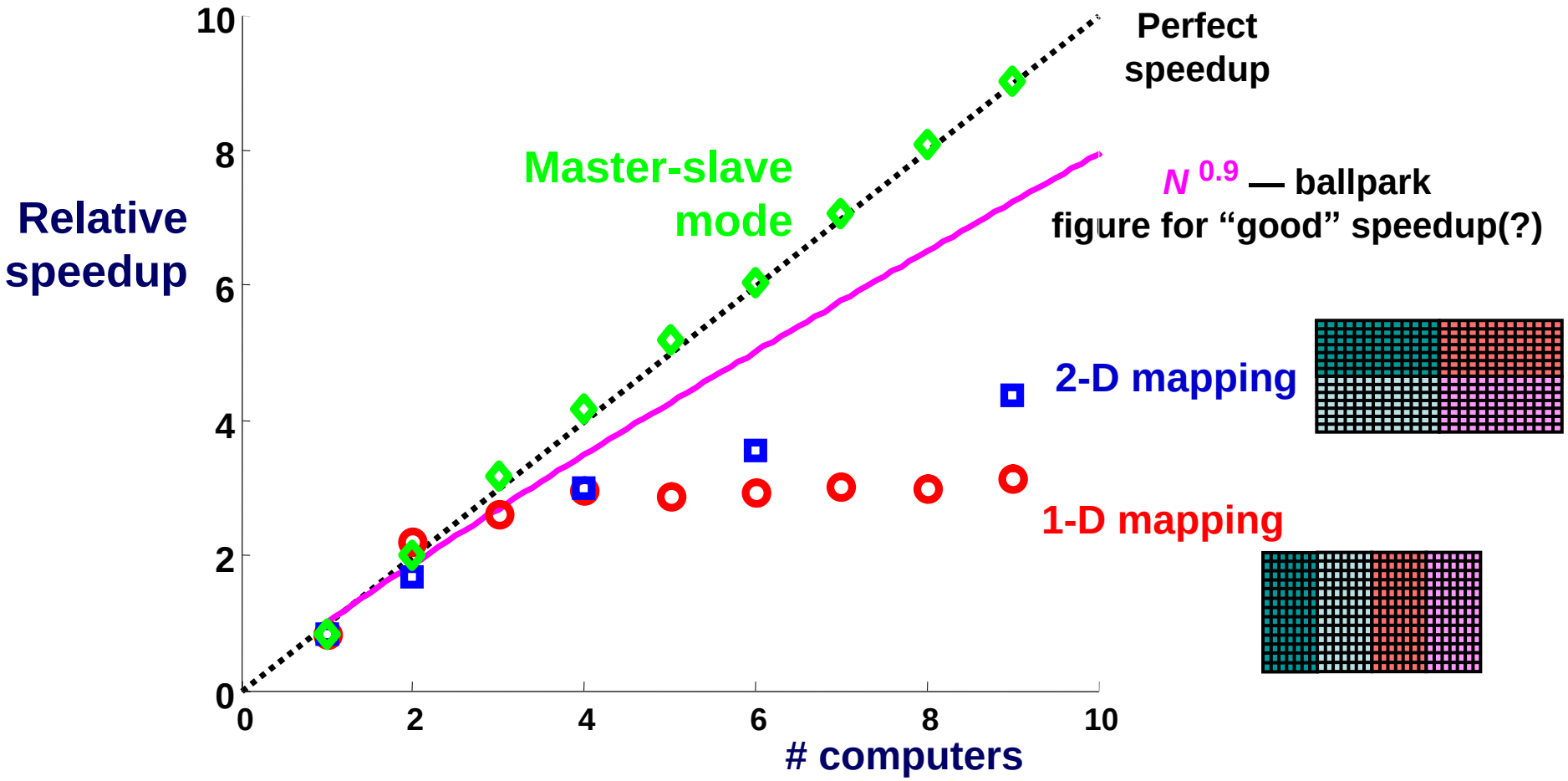
- each simulation fits on 1 machine, and
- I'm going to run multiple jobs anyway (optimization, band diagrams, etc.)



Advantages:

- no redundant grid points
- little network overhead
- less post-execution re-assembly of output files
- easier to combine heterogeneous machines

Parallelism is good.



Caveats:

- only 1-2 simulations may fit in the aggregate memory-space
- may need to see 1st answer before designing 2nd simulation

Conclusions

- **The FDTD (finite-difference time-domain) algorithm has many advantages as a numerical simulation tool for nanophotonics**
- **Its disadvantages all boil down to:**

"to be confident you will get the right answer, simulations must be large & slow"

•My motivation: to do *design optimization* with FDTD

need grid-computing to help do large, long simulations faster

•MPI implementation

- **supports all desired FDTD features**
- **demonstrated on up to 60 nodes**
- **for master-slave mode: 'perfect' scaling with # of computers**

•Future work

- **improve speedup for distributed problems**
- **JNI version for OptimalGrid**
- **get 'more' out of small FDTD simulations**