# Review: Why hardware matters and why it’s “parallel or bust”

- Architectural dependencies matter
  - Amdahl’s law: Speedup ≤ 1 / (Serial fraction)
  - Simple benchmarks exhibit complex, machine-specific behavior
- Physical limits
  - Processors are exploiting most available ILP
  - Little’s Law: latency * bandwidth = concurrency
  - Power ~ (no. of cores) * (frequency$^{2.5}$) and Perf ~ (no. cores) * (frequency)
  - Speed-of-light limit: ~ 1 Tflop/s with 1 TB memory on a 0.3 x 0.3 mm die

# From problem to parallel algorithm: An introductory example

Prof. Richard Vuduc

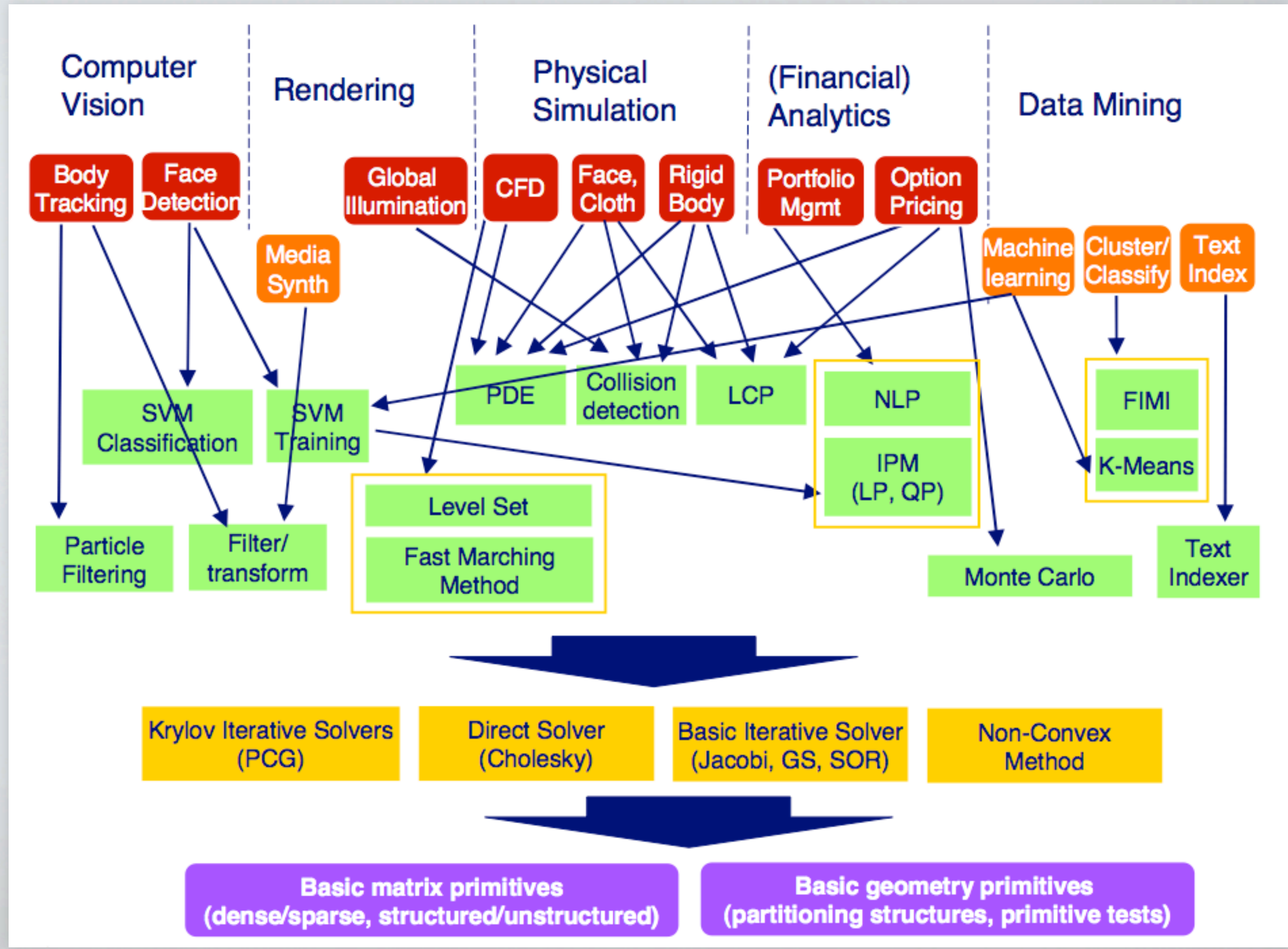
Georgia Institute of Technology

CSE/CS 8803 PNA, Spring 2008

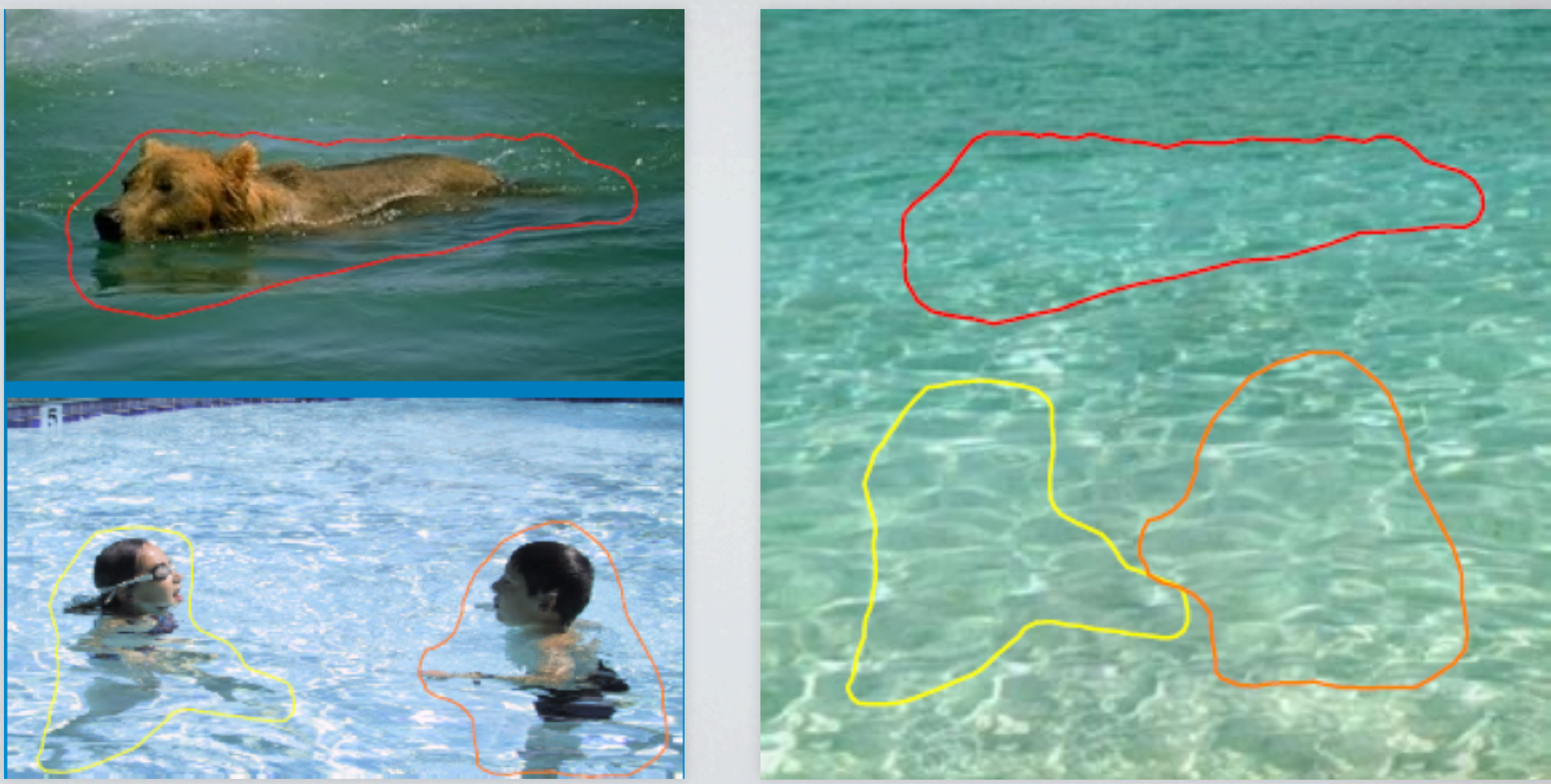
[L.03] Tuesday, January 15, 2008

# Sources for today's material

- CS 267 (Yelick & Demmel, UCB)
- "Sourcebook", eds. Dongarra, *et al*.
- Goldstein's book on classical mechanics
- Perez, *et al*. on Poisson image editing; Frédo Durand (MIT)
- Mike Heath at UIUC
- Michelle Strout's serial sparse tiling algorithm

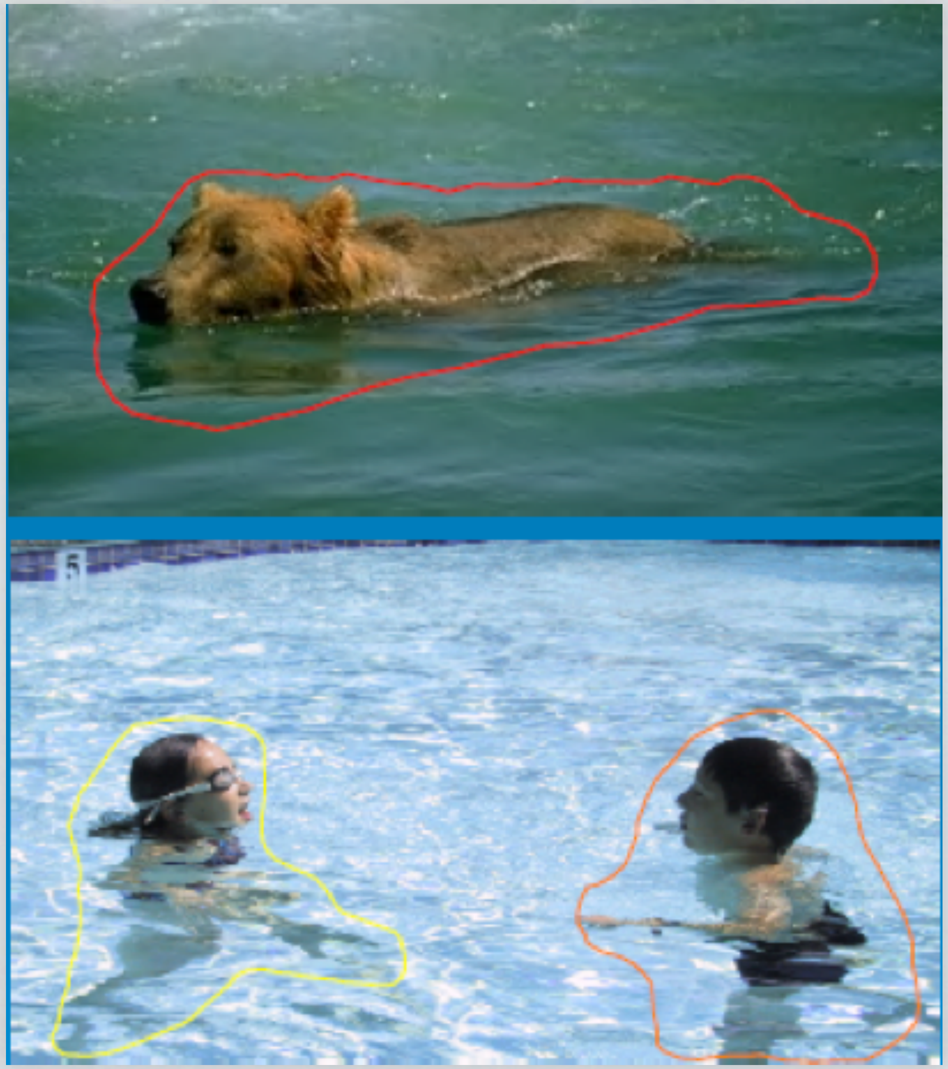



**Source: Dubey, *et al*., of Intel (2005)**



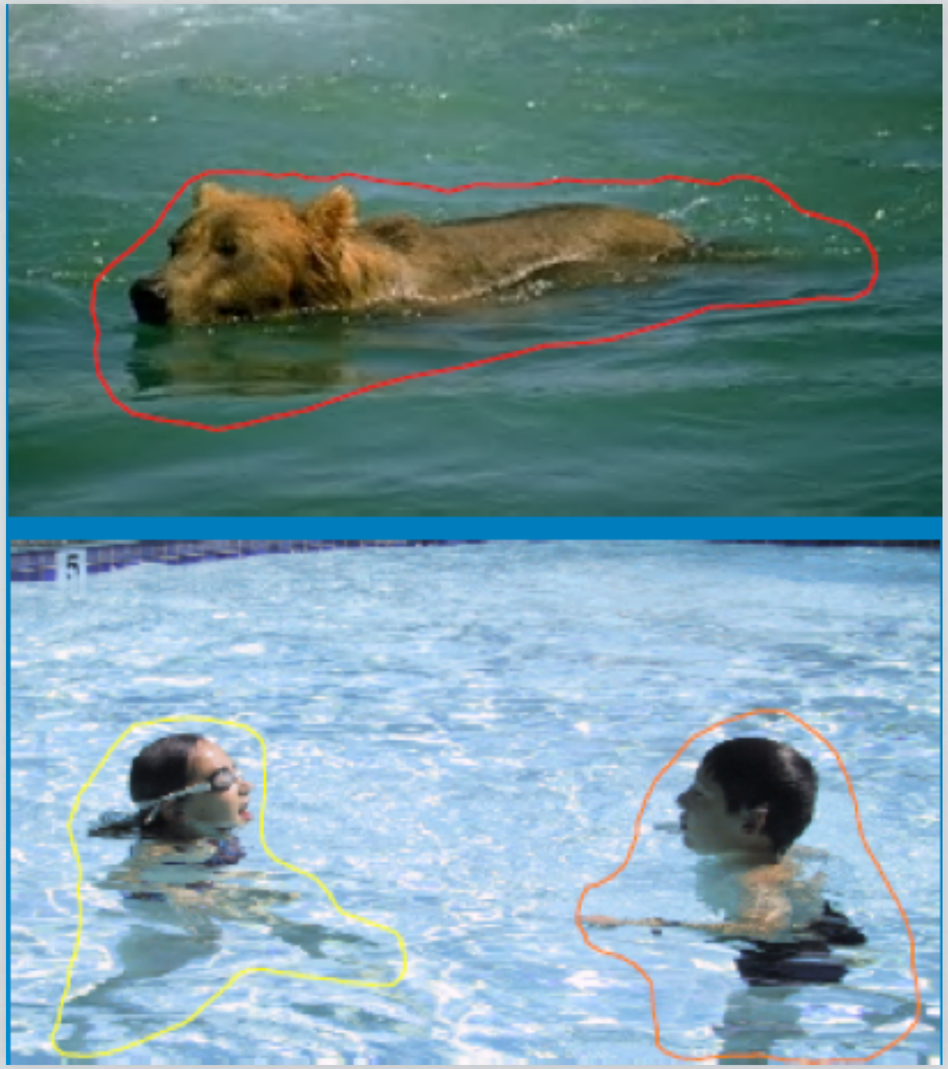
# Problem: Seamless image cloning.

(Source: Pérez, *et al.*, SIGGRAPH 2003)



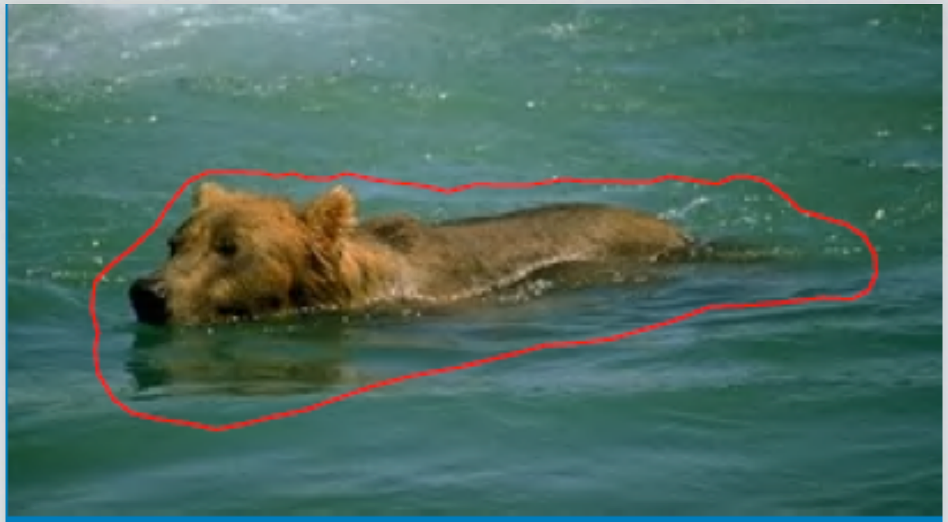
# Problem: Seamless image cloning.

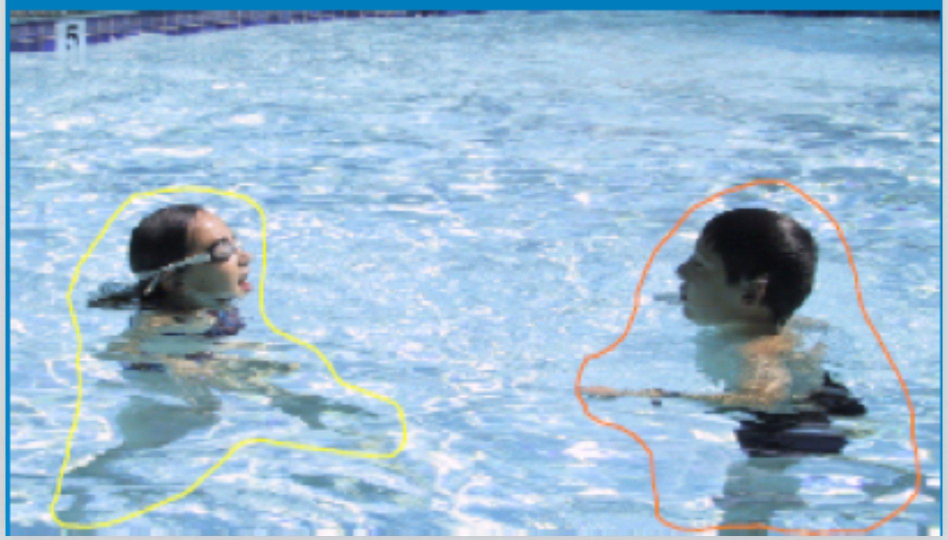
(Source: Pérez, *et al.*, SIGGRAPH 2003)



# Idea: Clone the *gradient*...

(Source: Pérez, *et al.*, SIGGRAPH 2003)





# … then reconstruct.

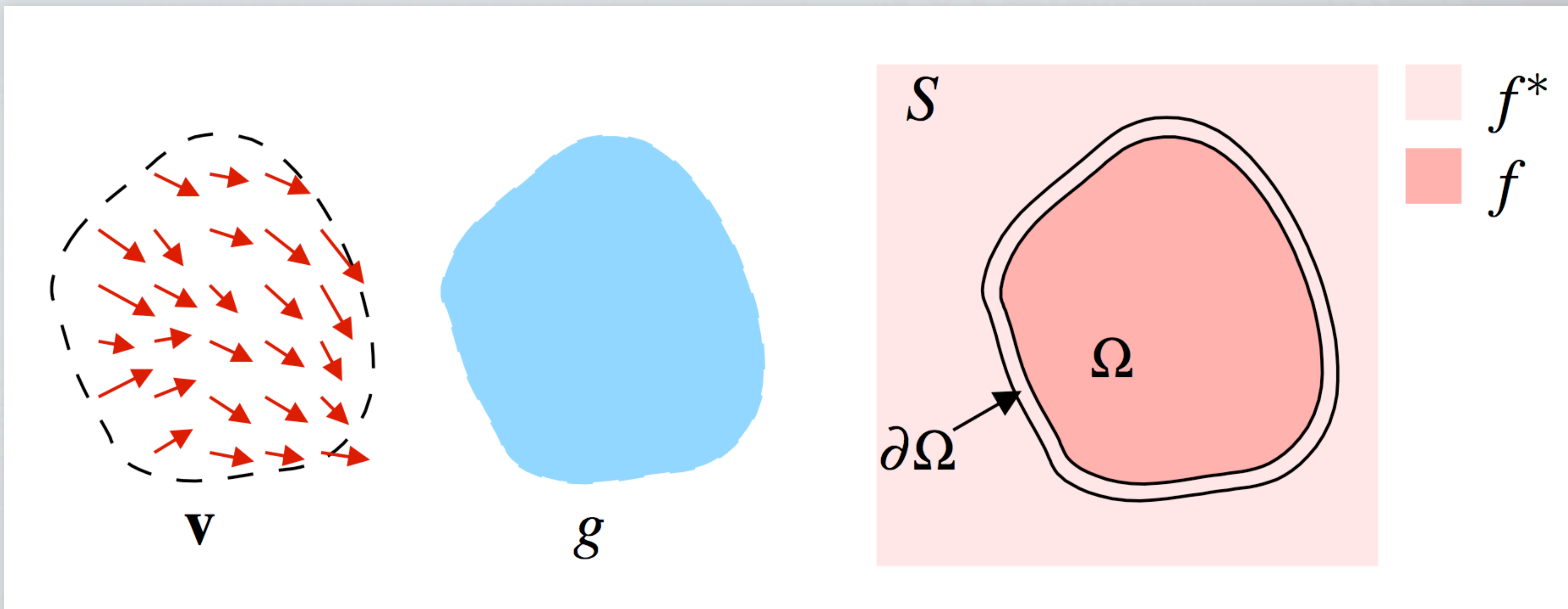
(Source: Pérez, *et al.*, SIGGRAPH 2003)

# Why the gradient?

Human visual system is sensitive to it; gradients encode edges well.

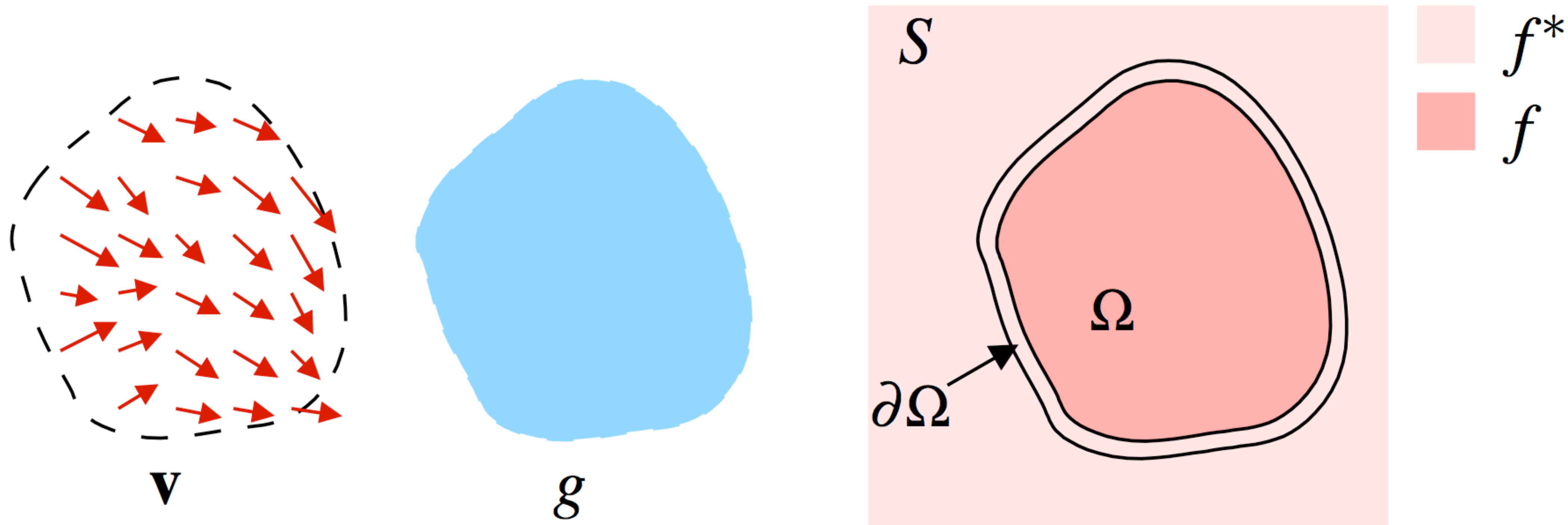
# Why the gradient?

Human visual system is sensitive to it; gradients encode edges well.



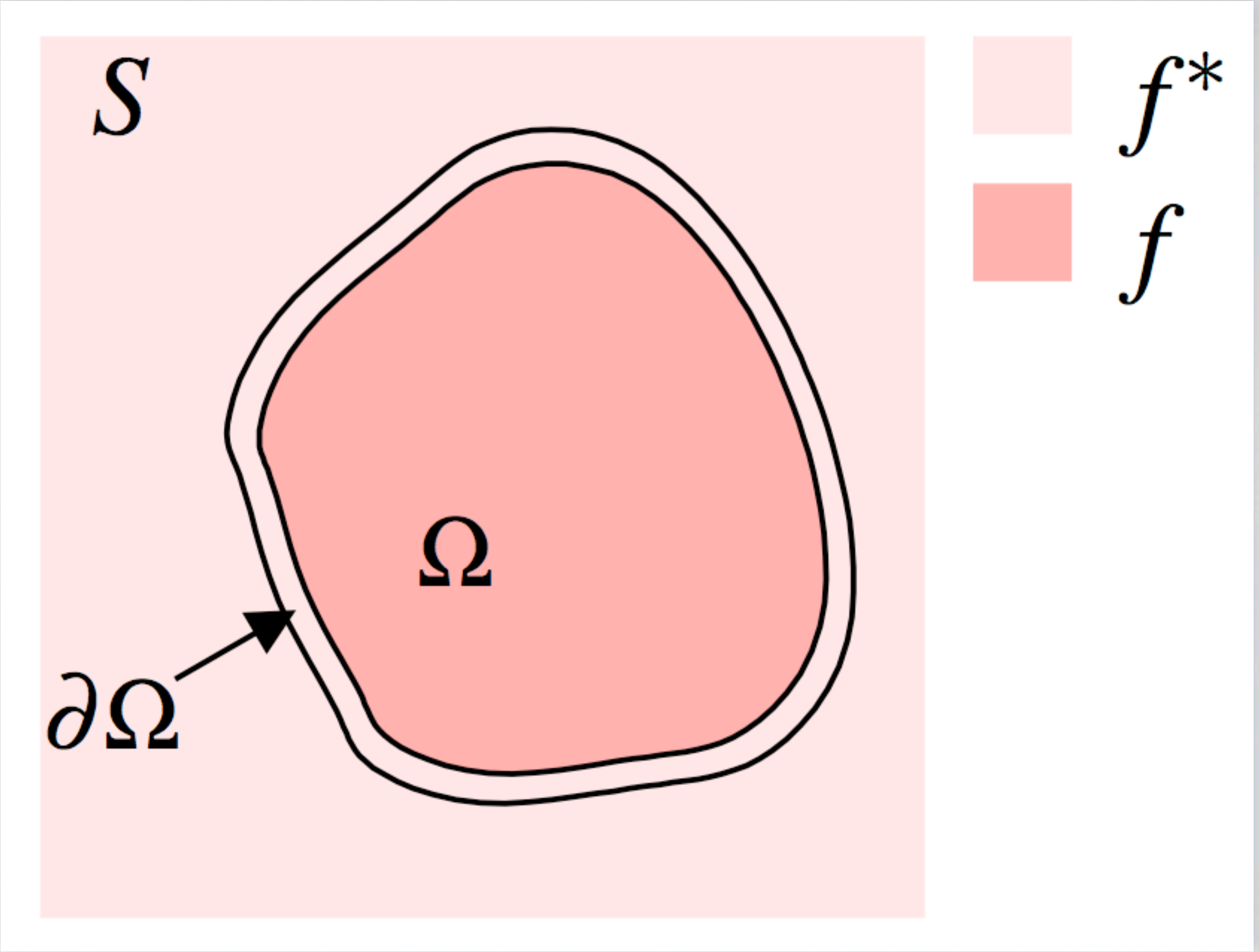

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Formulate as a “**guided interpolation**” problem.



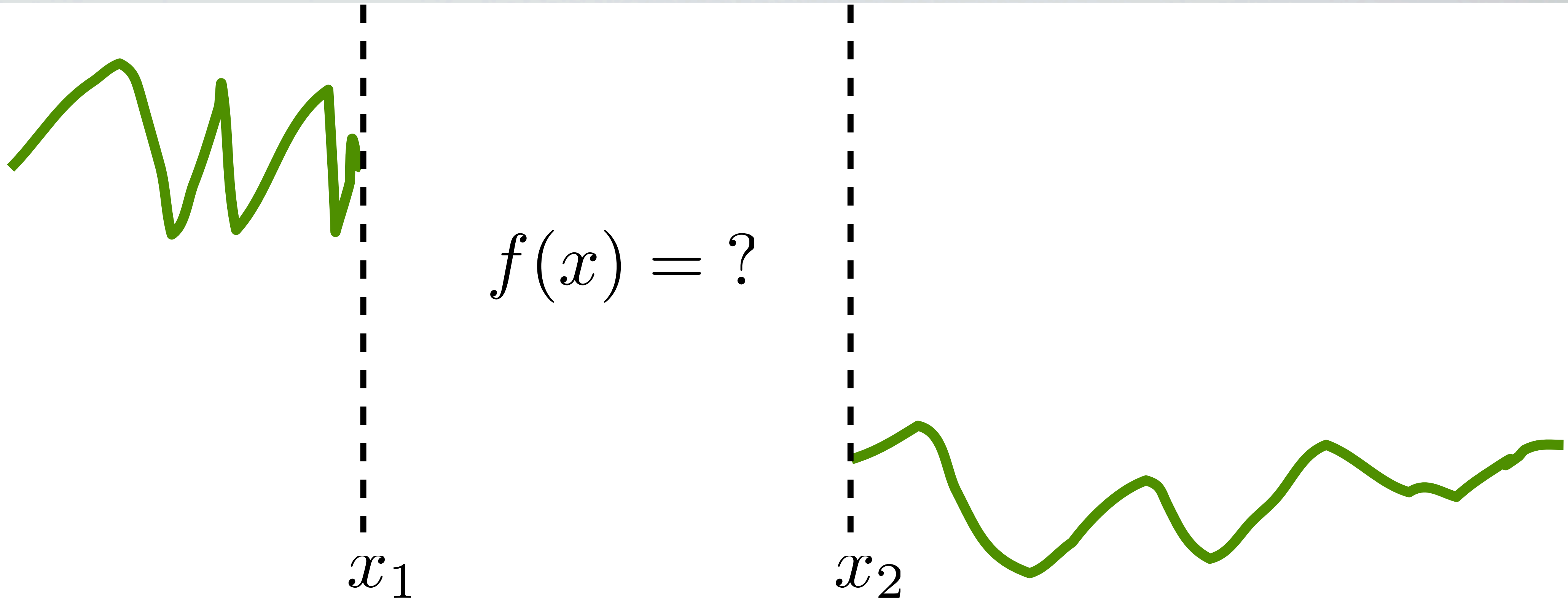

$$\nabla^2 f = \nabla \cdot \mathbf{v} \quad \text{over } \Omega \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Necessary condition: **Poisson's equation**.




$$\nabla^2 f = 0 \text{ over } \Omega \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

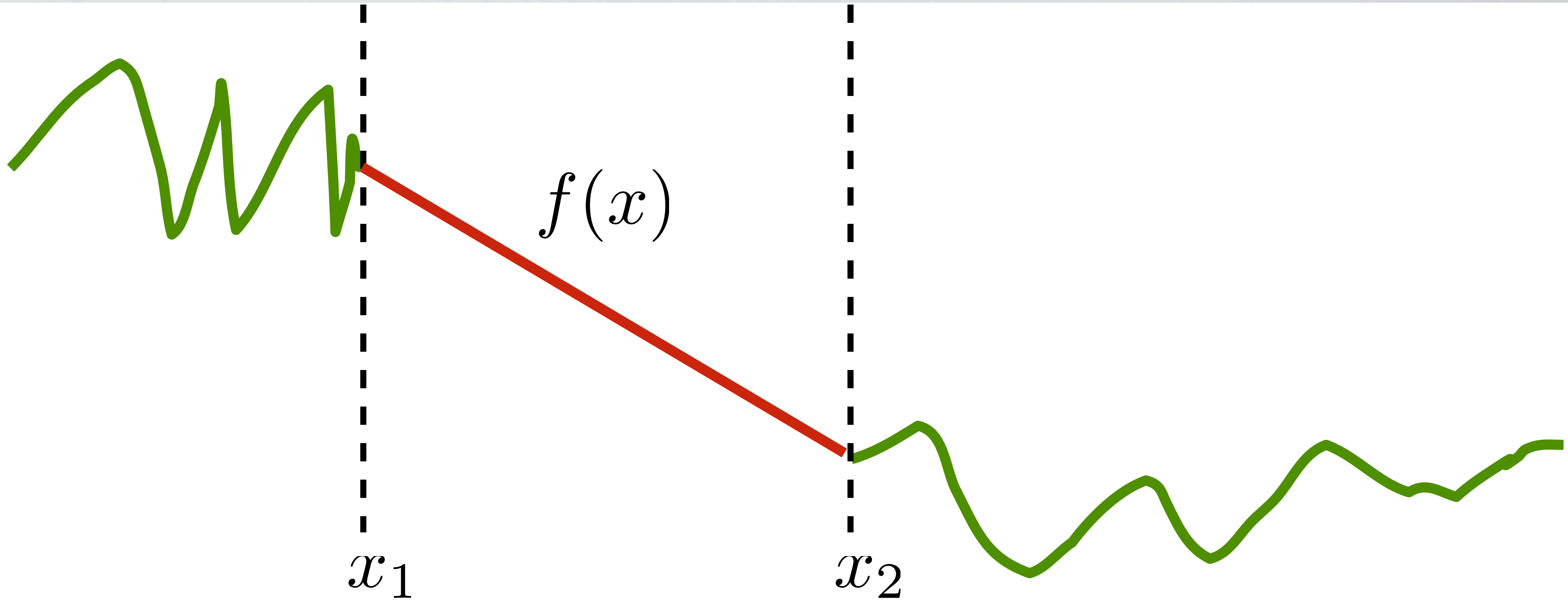
Simplest formulation: find the **membrane interpolant** ($\mathbf{v} = 0$).




$$\min_f \int_{x_1}^{x_2} (f'(x))^2 dx \quad \text{with} \quad f(x_1) = a \text{ and } f(x_2) = b$$

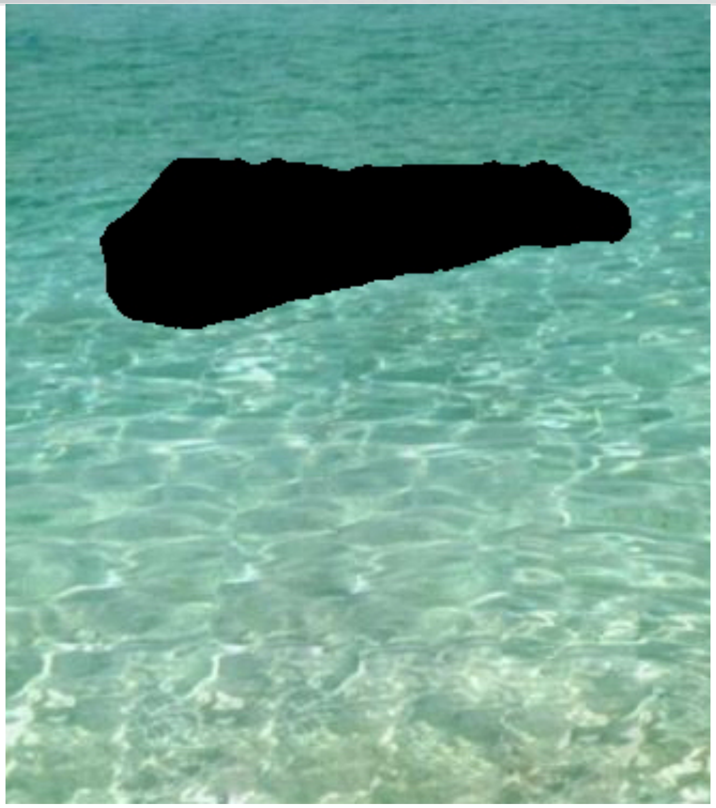
# Aside: Calculus of variations in 1-D

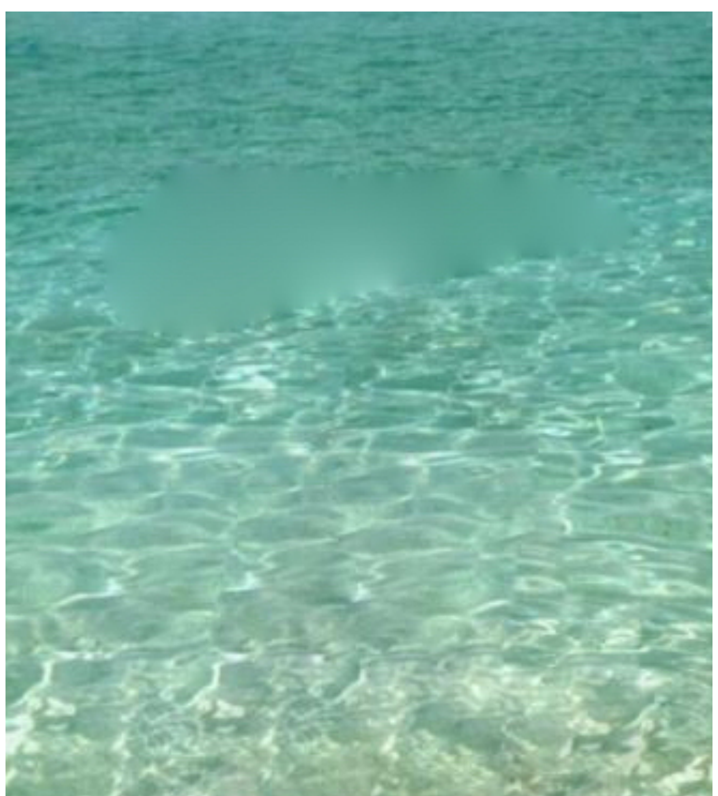
(see whiteboard)

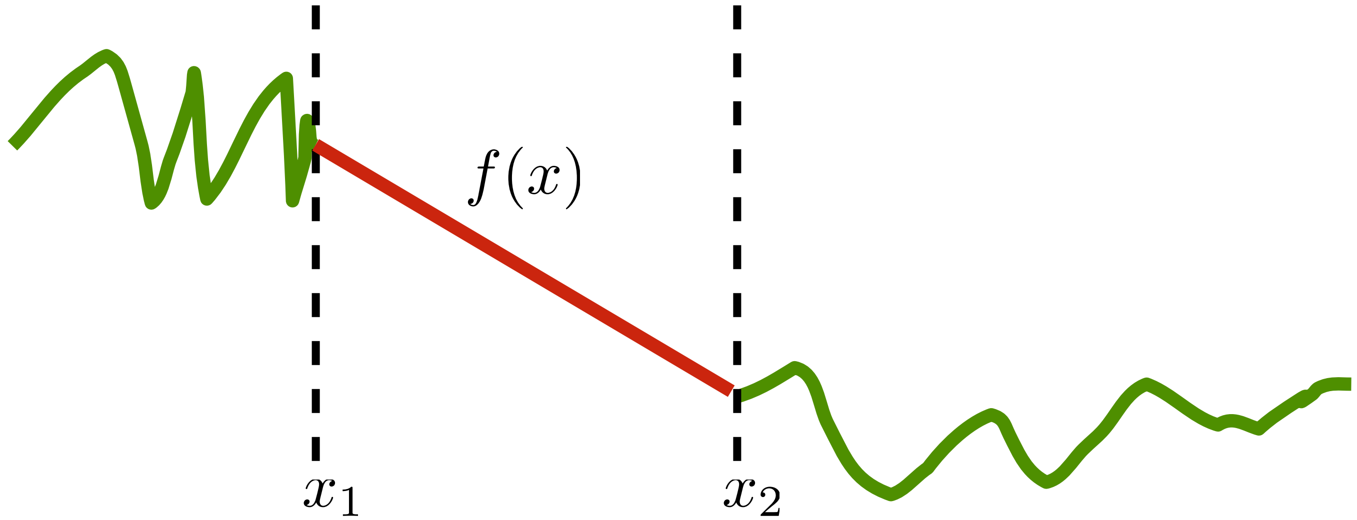


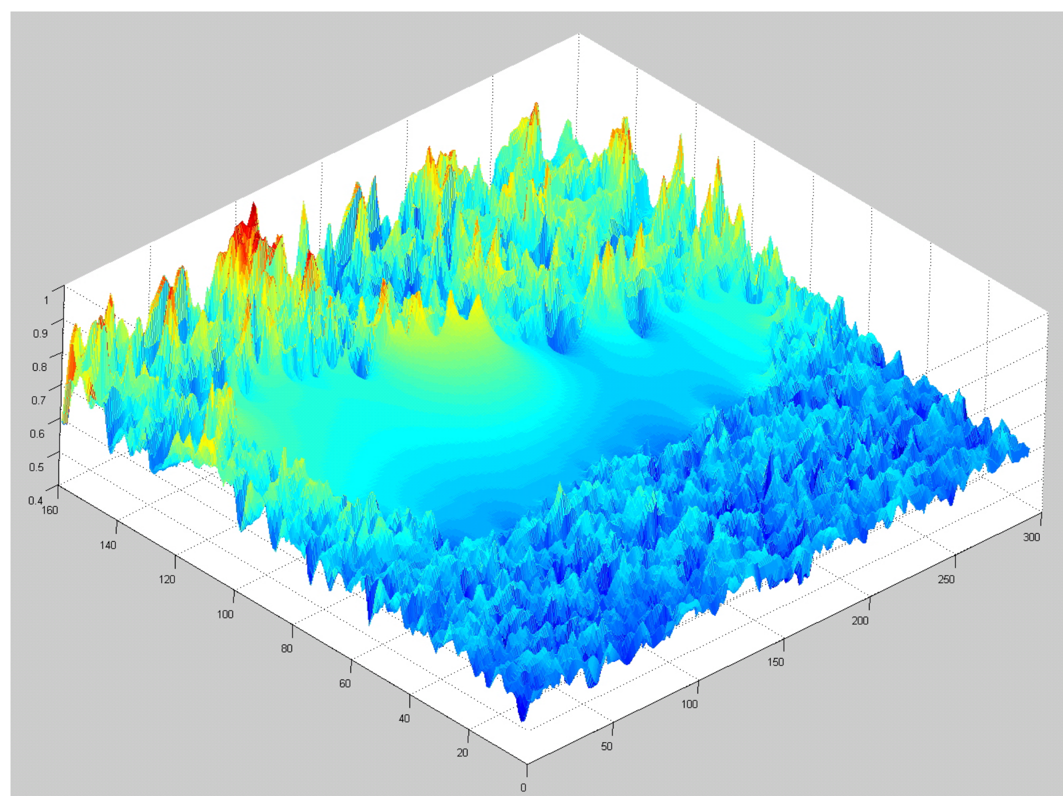

$$\min_f \int_{x_1}^{x_2} (f'(x))^2 dx \quad \text{s.t.} \quad f(x_1) = a \text{ and } f(x_2) = b$$

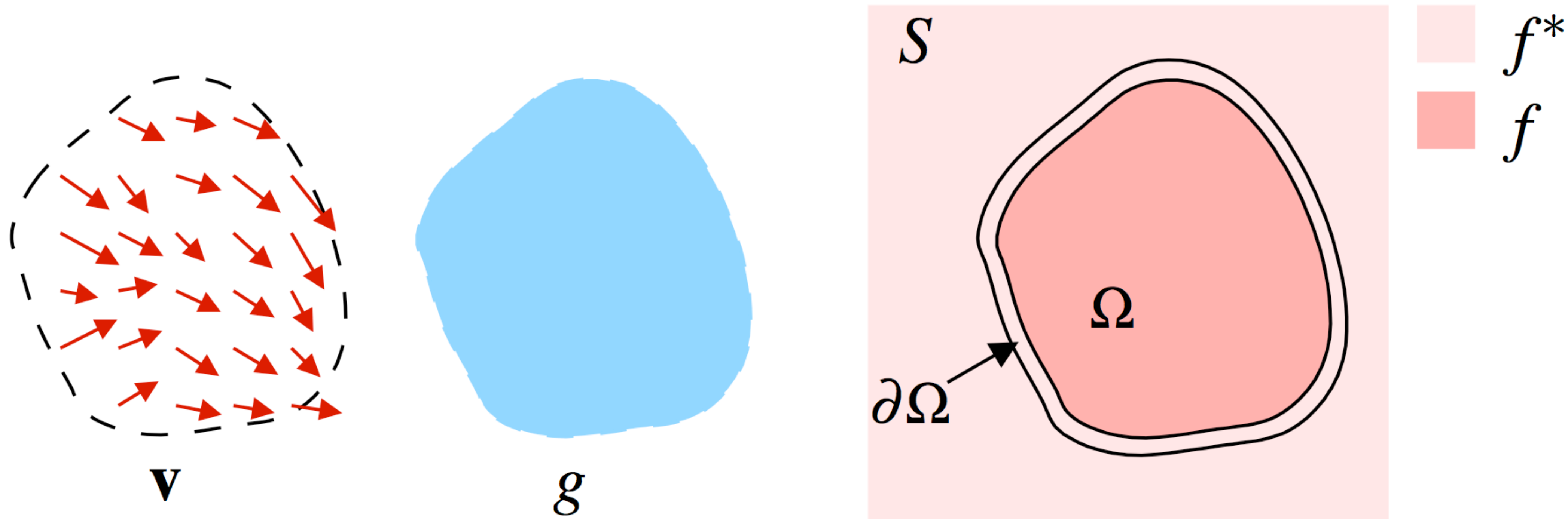
# Solution: Linear interpolation






$f(x)$
$x_1$
$x_2$






$$
\begin{aligned}
\mathbf{v} = \nabla g \quad &\Longrightarrow \quad \nabla^2 f = \nabla \cdot \nabla g = \nabla^2 g \\
&\Longleftrightarrow \quad \nabla^2 (f - g) = 0 \\
\text{Let } f = g + \hat{f} \quad &\Longrightarrow \quad \nabla^2 \hat{f} = 0, \ \text{ with } \hat{f}|_{\partial\Omega} = (f^* - g)|_{\partial\Omega}
\end{aligned}
$$

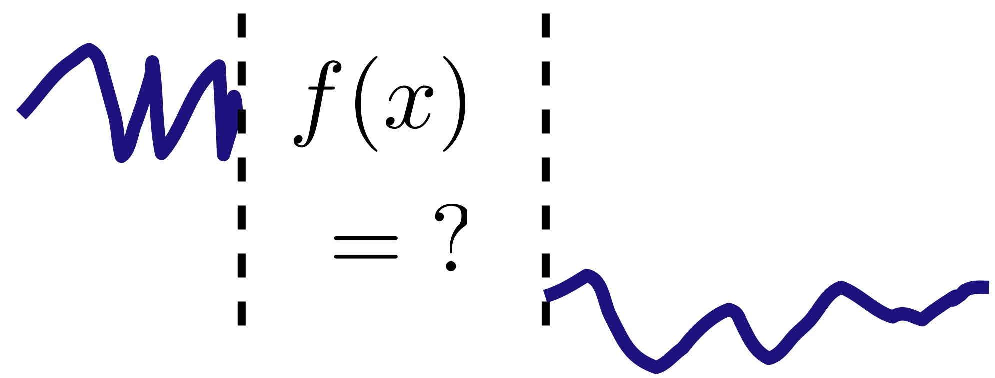
# 1-D analogue

# 1-D analogue

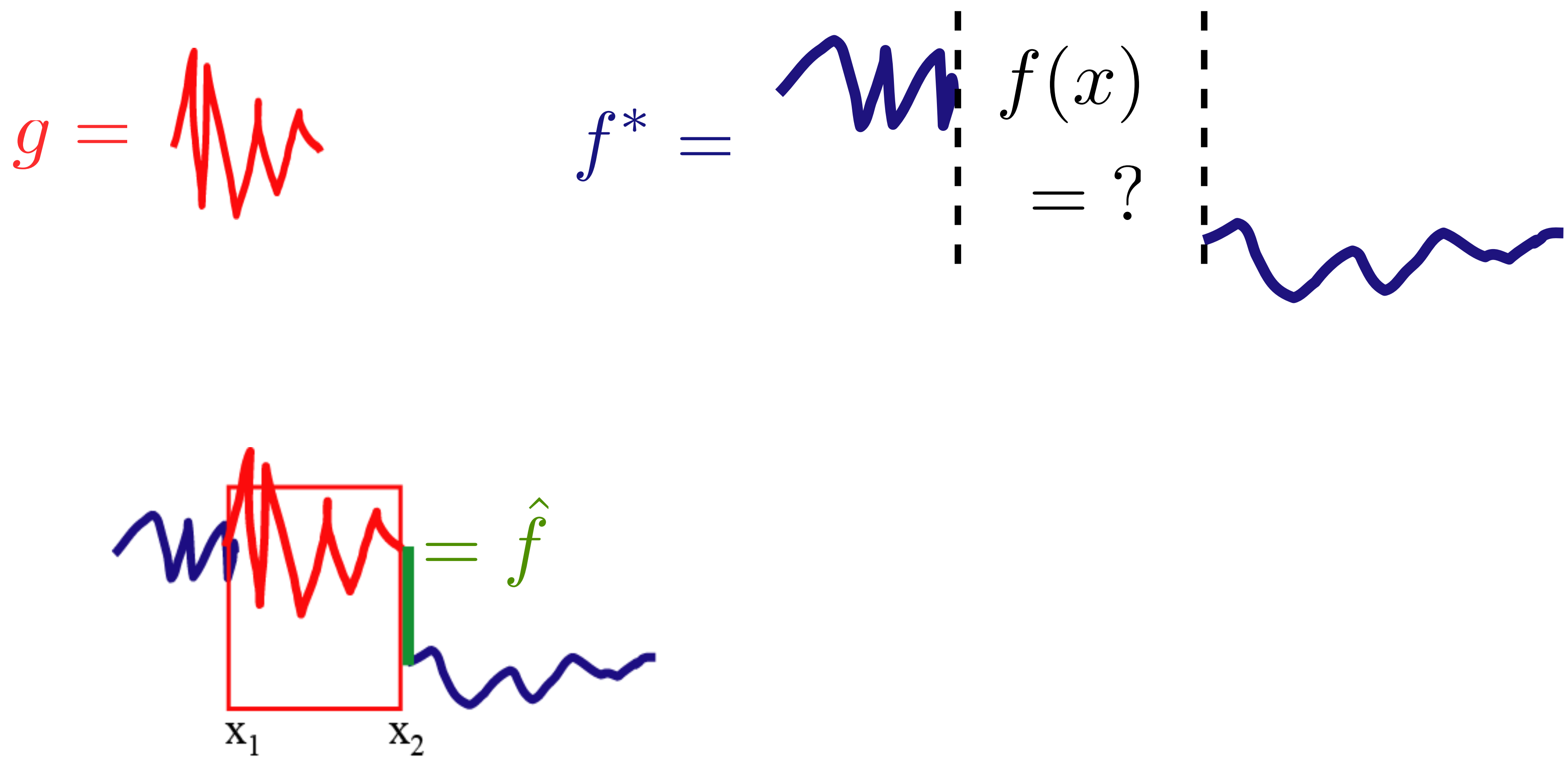
$g =$ 

# 1-D analogue

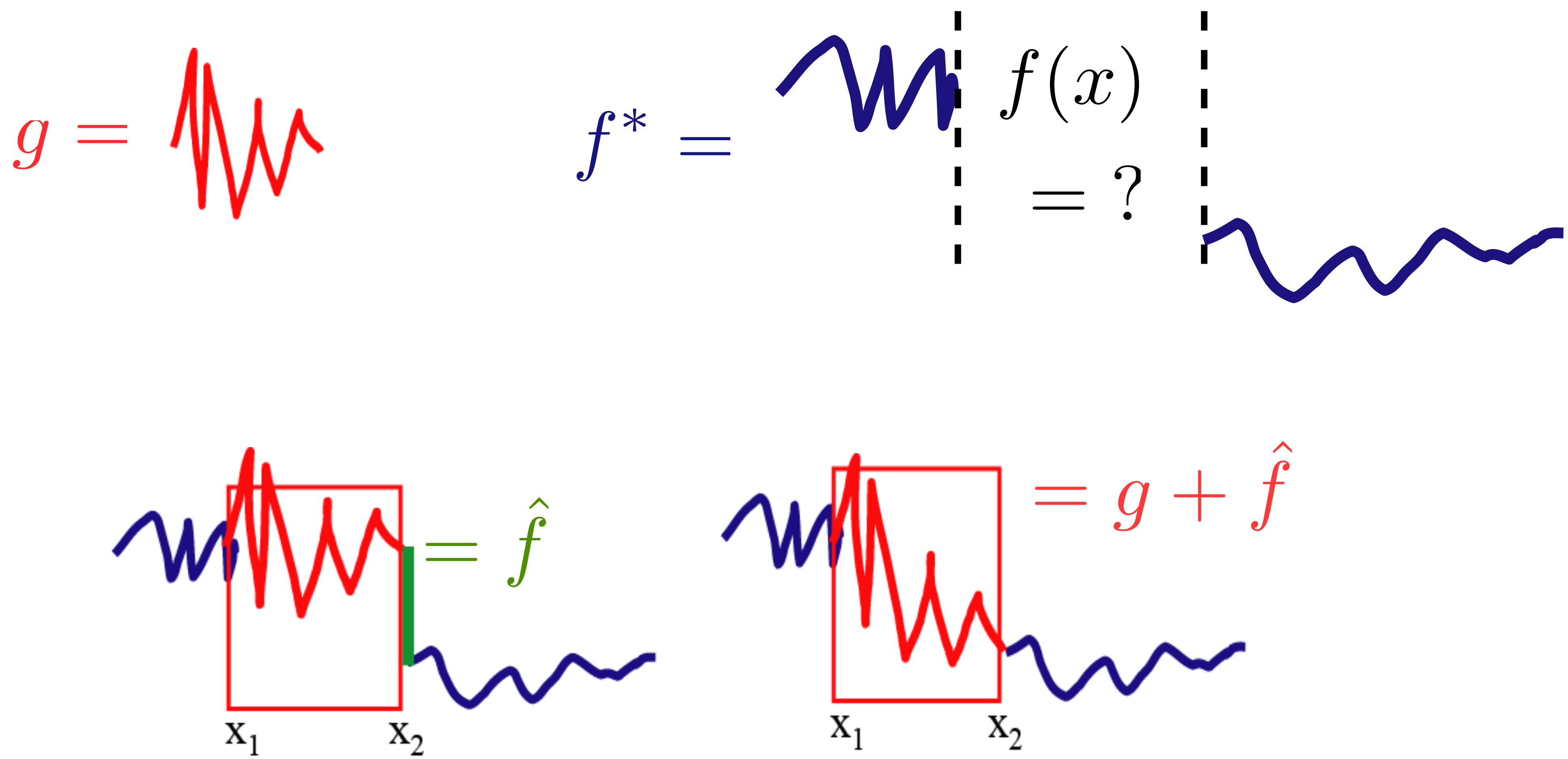
$g =$ 

$f^* =$ 

# 1-D analogue



# 1-D analogue



# Parallel numerical solutions of Poisson’s equation

# Examples of “traditional” applications of Poisson’s equation

- Electrostatics & gravitation: Potential = f(position)
- Heat flow: Temperature = f(position, time)
- Diffusion: Concentration = f(position, time)
- Fluid flow: Velocity, pressure, density = f(position, time)
- Elasticity: Stress, strain = f(position, time)

$$\text{(2-D) Find } u(x,y)\text{:} \qquad \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y)$$

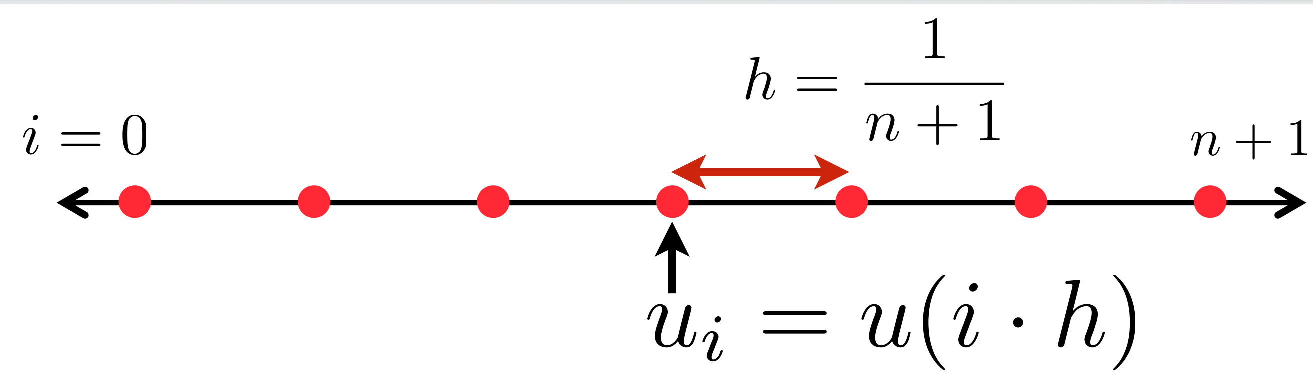
# Poisson's equation in 1-D:

$$-\frac{d^2u(x)}{dx^2} = f(x), \quad 0 < x < 1, \quad u(0) = u(1) = 0$$

# Poisson's equation in 1-D:

$$-\frac{d^2u(x)}{dx^2} = f(x), \quad 0 < x < 1, \quad u(0) = u(1) = 0$$

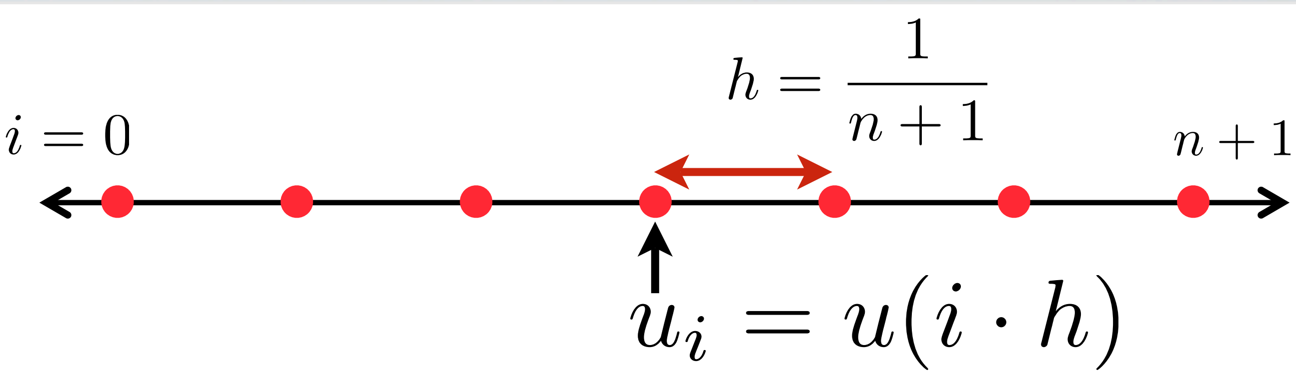
**Discretize:**



# Poisson's equation in 1-D:

$$-\frac{d^2u(x)}{dx^2} = f(x), \quad 0 < x < 1, \quad u(0) = u(1) = 0$$

**Discretize:**



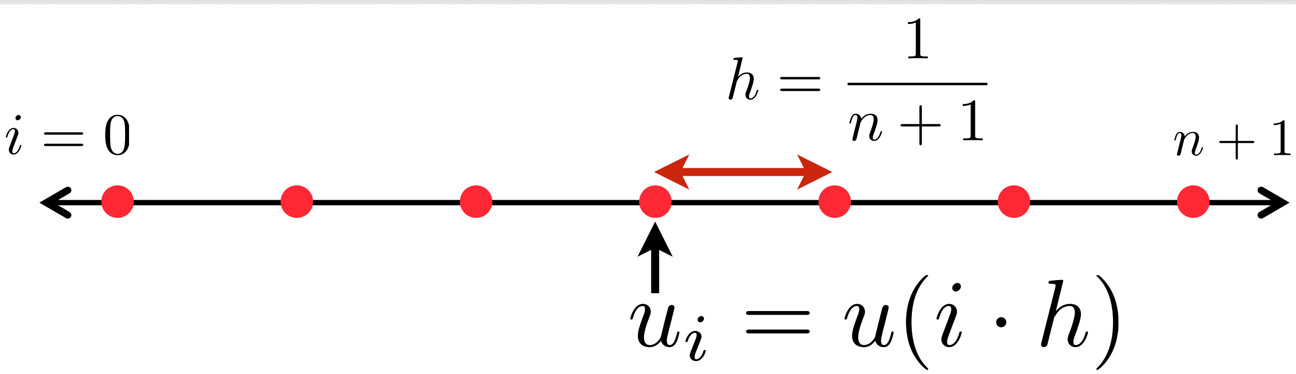

**Approximate:**

$$-\frac{d^2u(x)}{dx^2}\Big|_{x=x_i} \approx \frac{2u_i - u_{i-1} - u_{i+1}}{h^2}$$

# Poisson's equation in 1-D:

$$-\frac{d^2u(x)}{dx^2} = f(x), \quad 0 < x < 1, \quad u(0) = u(1) = 0$$

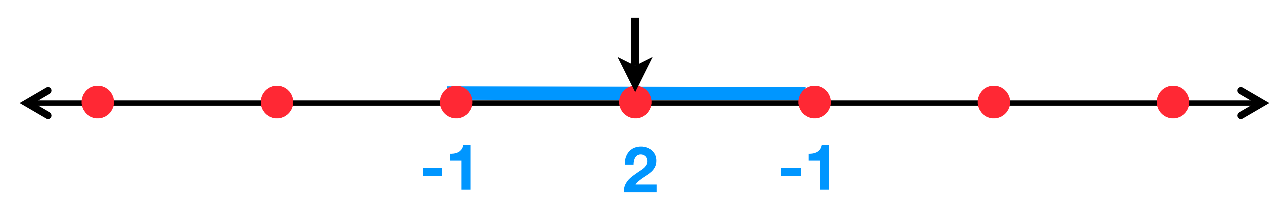
**Discretize:**




**Approximate:**

$$-\frac{d^2u(x)}{dx^2}\Big|_{x=x_i} \approx \frac{2u_i - u_{i-1} - u_{i+1}}{h^2}$$

**"Stencil":**



# Express stencil in matrix notation

**Approximation:** $\approx \dfrac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} = f_i \triangleq f(ih)$

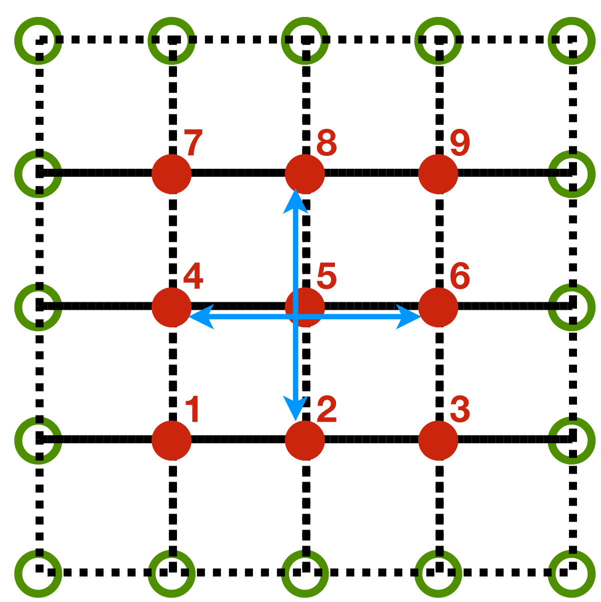
$$\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \cdots & & \\ & & & -1 & 2 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{pmatrix} = -h^2 \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{pmatrix}$$

$$\Downarrow$$

$$T \cdot u = -h^2 f$$

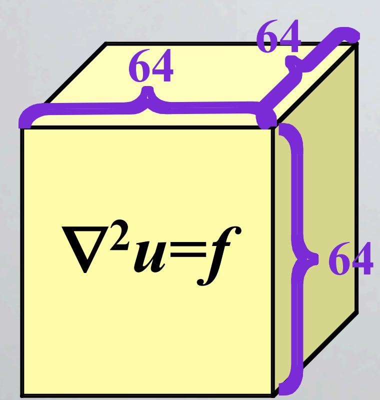
# 2-D Poisson Equation

**Graph and stencil**




$$T = \begin{array}{c} \begin{array}{ccccccccc} & 2 & & 4 & 5 & 6 & & 8 & \end{array} \\ \left(\begin{array}{ccc|ccc|ccc} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ \hline -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ \hline & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{array}\right) \end{array}$$

| Year | Method | Reference | Storage | Flops |
|---|---|---|---|---|
| 1947 | Gaussian Elimination | Von Neumann & Goldstine | $n^5$ | $n^7$ |
| 1950 | Optimal SOR | Young | $n^3$ | $n^4 \log n$ |
| 1971 | Conj. grad. | Reid | $n^3$ | $n^{3.5} \log n$ |
| 1977 | FFT | Pickering | $n^3$ | $n^3 \log n$ |
| 1984 | Full multigrid | Brandt | $n^3$ | $n^3$ |




# Recall: Poisson solution methods.

If n=64, flops reduced by ~16 M [6 mo. to 1 sec.]; Source: Keyes (2004)

## Algorithms for 2-D (3-D) Poisson, $N=n^2$ ($=n^3$)

| Algorithm | Serial | PRAM | Memory | # procs |
|---|---|---|---|---|
| *Dense LU* | $N^3$ | $N$ | $N^2$ | $N^2$ |
| *Band LU* | $N^2$ ($N^{7/3}$) | $N$ | $N^{3/2}$ ($N^{5/3}$) | $N$ ($N^{4/3}$) |
| Jacobi | $N^2$ ($N^{5/3}$) | $N$ ($N^{2/3}$) | $N$ | $N$ |
| *Explicit inverse* | $N^2$ | $\log N$ | $N^2$ | $N^2$ |
| Conj. grad. | $N^{3/2}$ ($N^{4/3}$) | $N^{1/2(1/3)} \log N$ | $N$ | $N$ |
| RB SOR | $N^{3/2}$ ($N^{4/3}$) | $N^{1/2}$ ($N^{1/3}$) | $N$ | $N$ |
| *Sparse LU* | $N^{3/2}$ ($N^2$) | $N^{1/2}$ | $N \log N$ ($N^{4/3}$) | $N$ |
| *FFT* | $N \log N$ | $\log N$ | $N$ | $N$ |
| Multigrid | $N$ | $\log^2 N$ | $N$ | $N$ |
| ***Lower bound*** | **$N$** | **$\log N$** | **$N$** | |

**PRAM** = idealized parallel model with zero communication cost.
*Source: Demmel (1997)*

# Cost may depend on problem properties

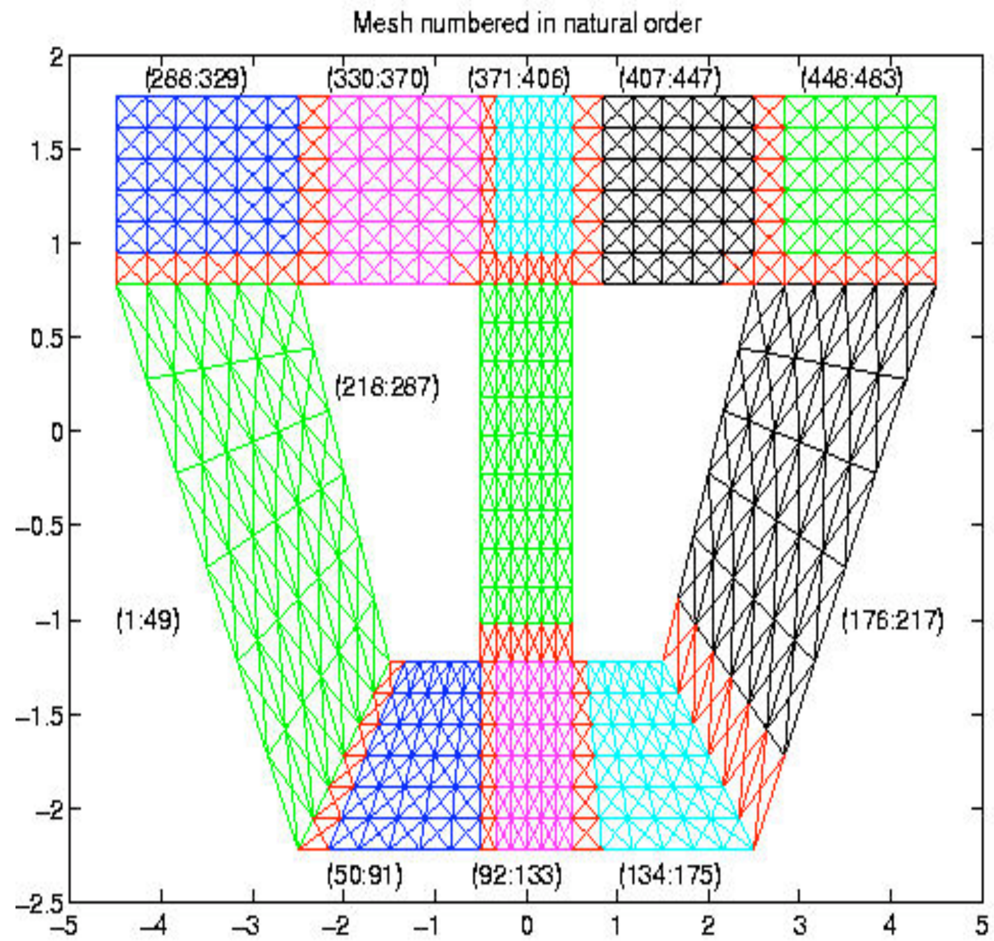
| Alg | Cost |
|---|---|
| Dense LU | $N^3$ ($p=N^2 \Rightarrow N$) |
| Band LU | $N*b^2$ ($p=b^2 \Rightarrow N$) |
| Jacobi | Cost(SpMV) * (# its), where t = k |
| CG | (SpMV + dot) * (# its), where # its = sqrt(k) |
| RB SOR | (SpMV) * (# its), where #its = sqrt(k) |

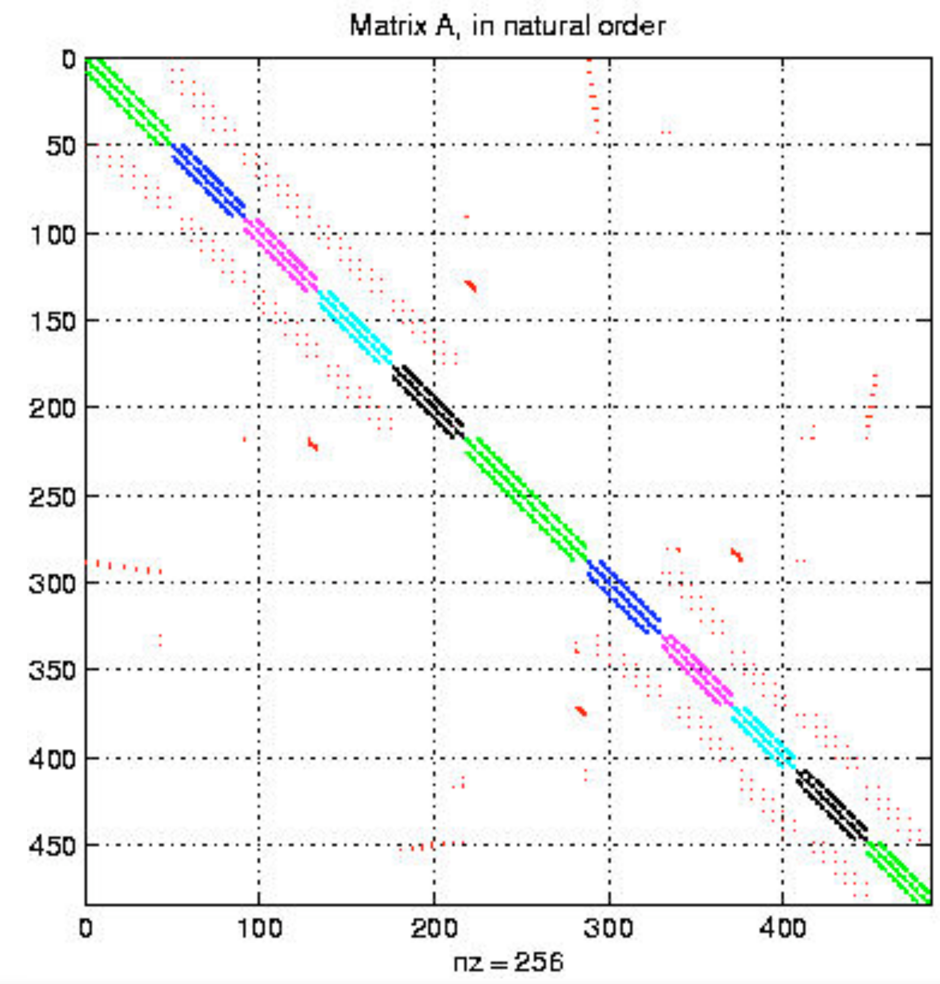
- Dim. = $n^2$ ($n^3$)
- Bandwidth b = n ($n^2$)
- Condition number k = $n^2$ (same)
- No. iterations = t
- SpMV = sparse matrix-vector multiply

# "Practical" meshes

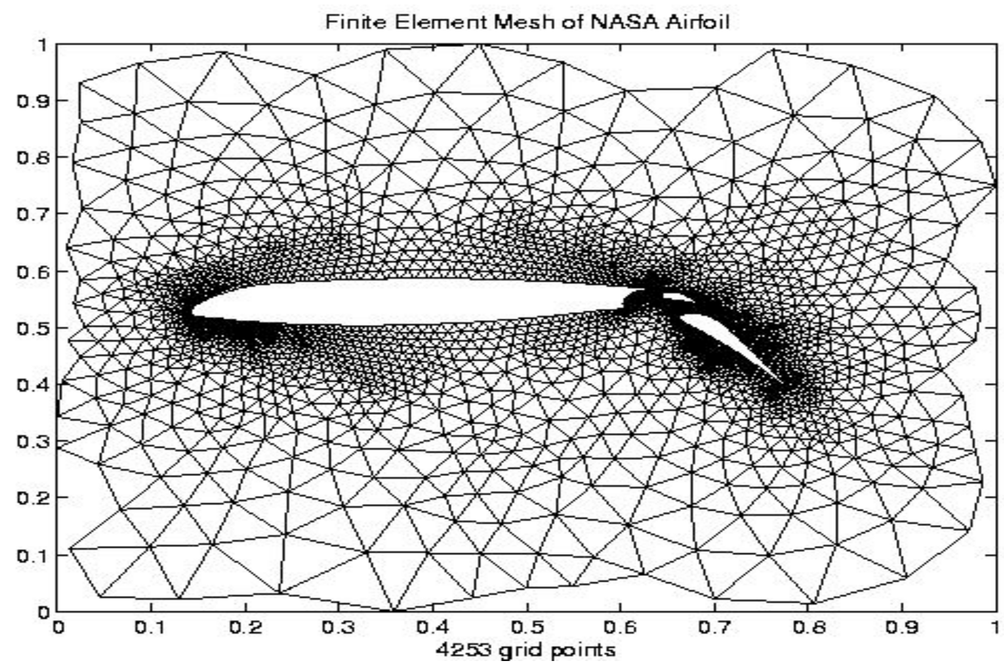
- Regular 1-D, 2-D, and 3-D are building blocks
- Practical meshes are irregular
  - **Composite**: Stitch regular meshes
  - **Unstructured**: Arbitrary mesh points and connectivity
  - **Adaptive**: Change during solve

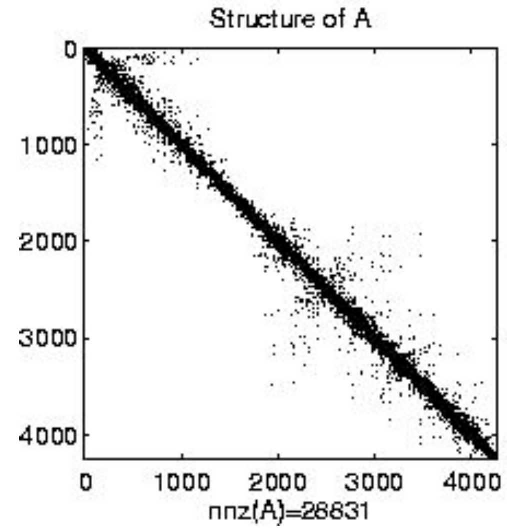
# Example mesh: Mechanical structure

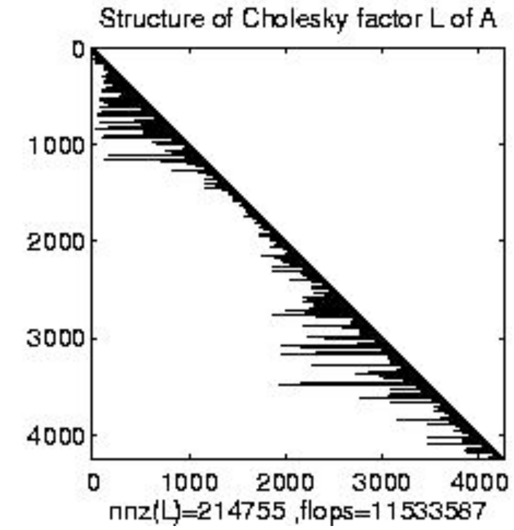


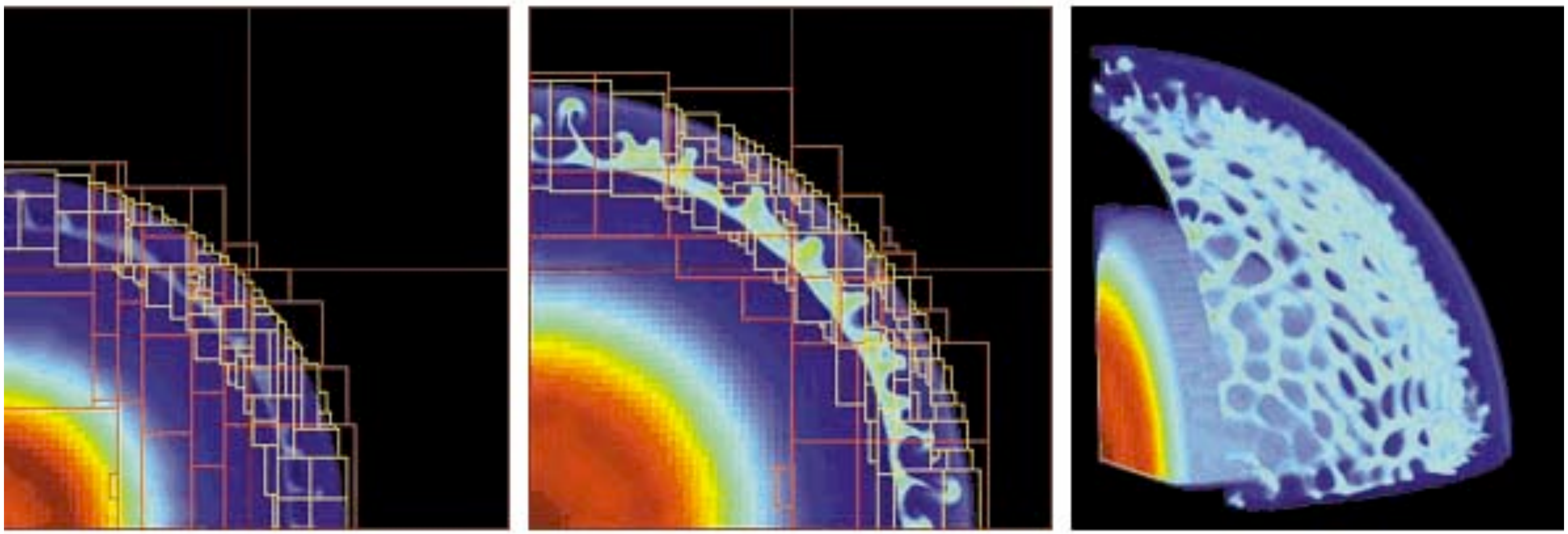
# Example mesh: NASA airfoil



# Example mesh:
# Adaptive mesh refinement (AMR)



Mesh refined near fine-grained behavior.
*Source: Bell & Colella (LBNL)*

# Administrivia

# Administrivia

- Need accounts? Send me an e-mail. HW 1 goes out 1/24
- Old T-square site partly restored; resubmit HW 0 (wait for mail from me)
- Project proposals “assigned,” due ~ 7-8th week
- Auditors?

# Serial and parallel Jacobi

## Algorithms for 2-D (3-D) Poisson, $N=n^2$ ($=n^3$)

| Algorithm | Serial | PRAM | Memory | # procs |
|---|---|---|---|---|
| *Dense LU* | $N^3$ | $N$ | $N^2$ | $N^2$ |
| *Band LU* | $N^2$ ($N^{7/3}$) | $N$ | $N^{3/2}$ ($N^{5/3}$) | $N$ ($N^{4/3}$) |
| **Jacobi** | **$N^2$ ($N^{5/3}$)** | **$N$ ($N^{2/3}$)** | **$N$** | **$N$** |
| *Explicit inverse* | $N^2$ | $\log N$ | $N^2$ | $N^2$ |
| Conj. grad. | $N^{3/2}$ ($N^{4/3}$) | $N^{1/2(1/3)} \log N$ | $N$ | $N$ |
| RB SOR | $N^{3/2}$ ($N^{4/3}$) | $N^{1/2}$ ($N^{1/3}$) | $N$ | $N$ |
| *Sparse LU* | $N^{3/2}$ ($N^2$) | $N^{1/2}$ | $N \log N$ ($N^{4/3}$) | $N$ |
| *FFT* | $N \log N$ | $\log N$ | $N$ | $N$ |
| Multigrid | $N$ | $\log^2 N$ | $N$ | $N$ |
| ***Lower bound*** | **$N$** | **$\log N$** | **$N$** | |

**PRAM** = idealized parallel model with zero communication cost.

*Source: Demmel (1997)*

# Jacobi's method

- Rearrange terms in (2-D) Poisson:

$$u_{i,j} = \frac{1}{4}\left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} + h^2 f_{i,j}\right)$$

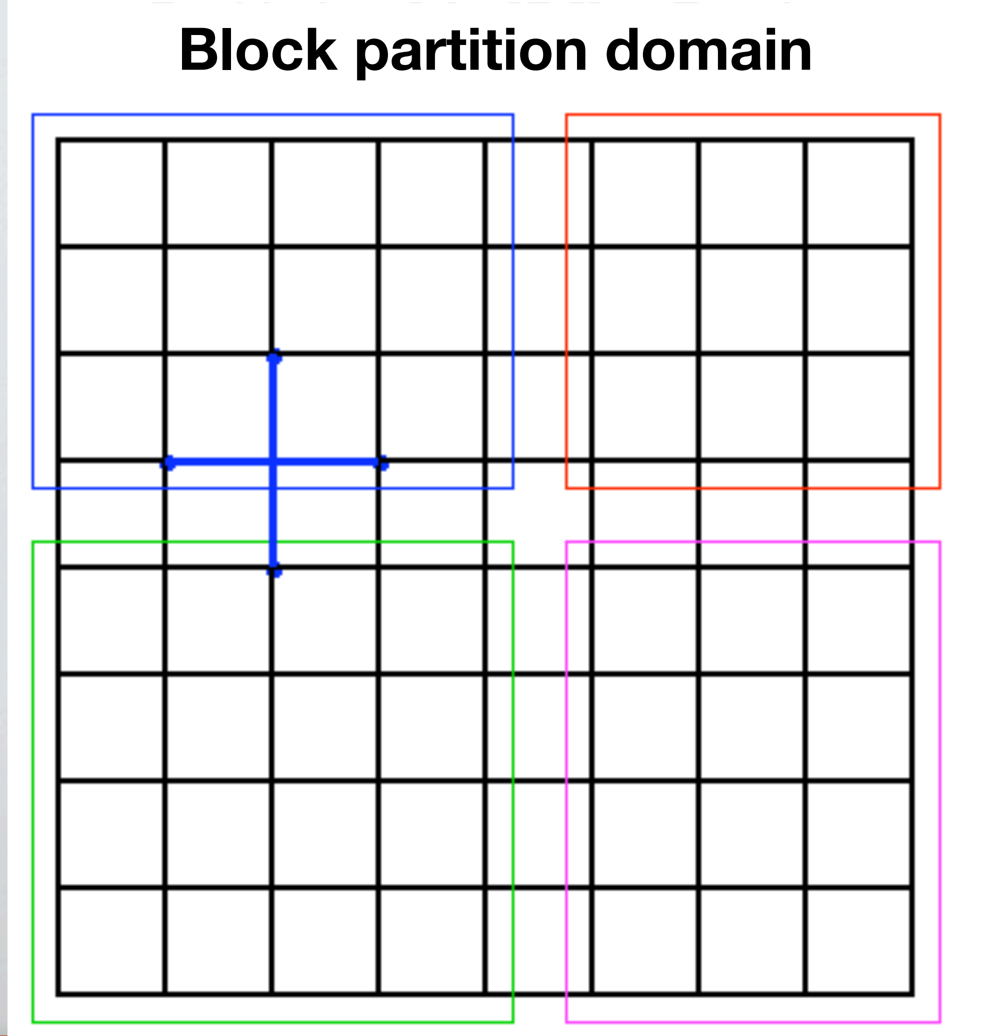
- For each (i, j), iteratively update (weighted averaging):

$$u_{i,j}^{t+1} = \frac{1}{4}\left(u_{i-1,j}^{t} + u_{i+1,j}^{t} + u_{i,j-1}^{t} + u_{i,j+1}^{t} + h^2 f_{i,j}\right)$$

- Motivation: Make solution at each point match discrete Poisson exactly.

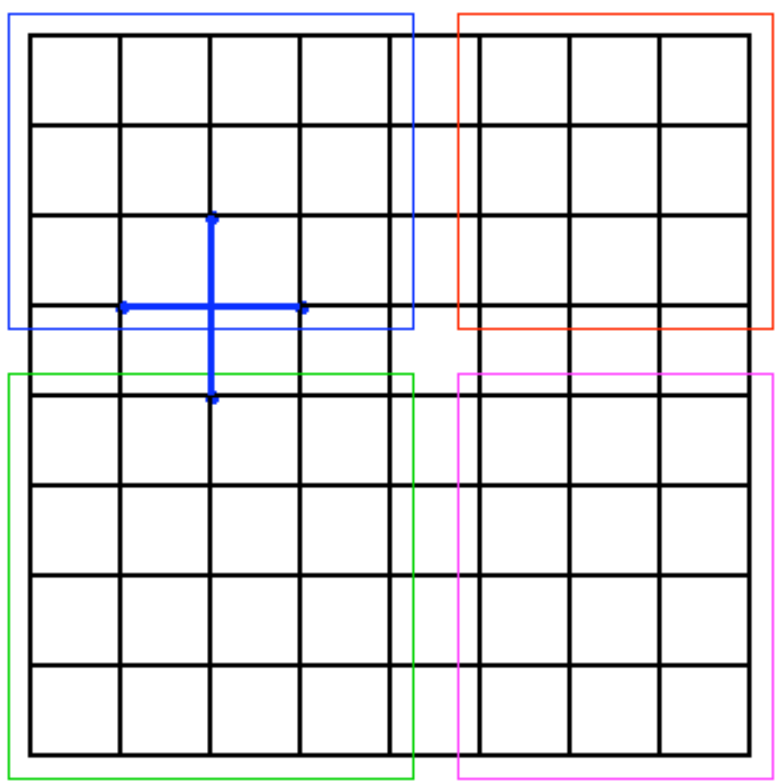
# Jacobi's method is easy to parallelize

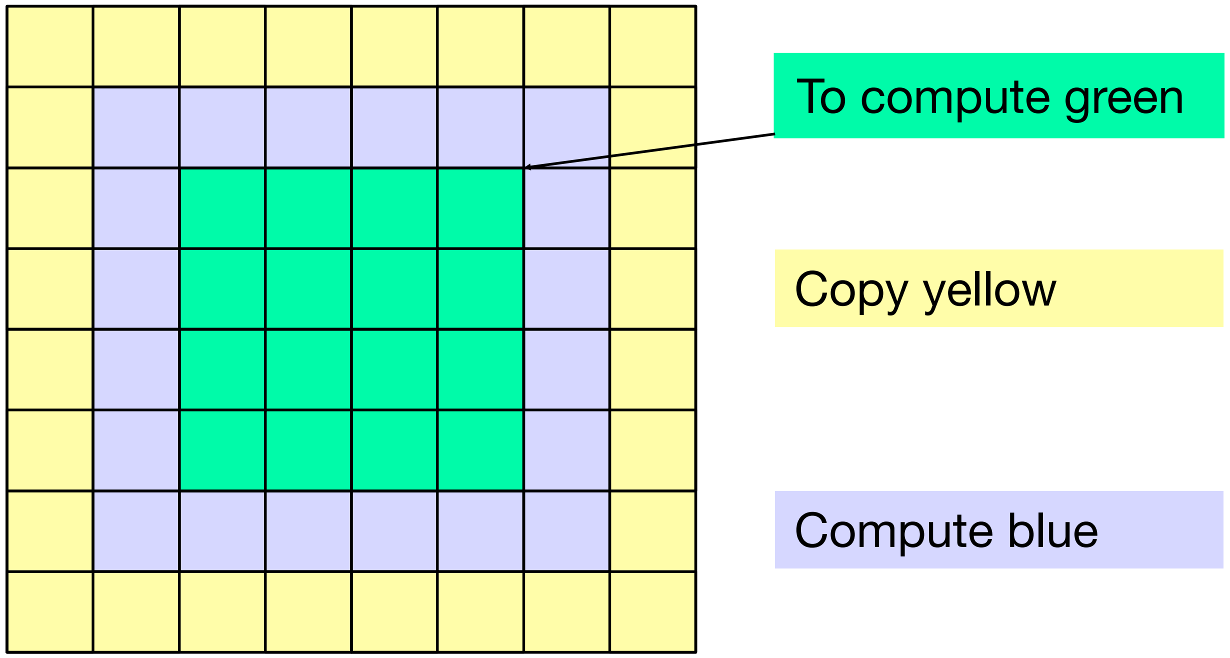
- Parallelism: Update all points independently
- Partition domain into blocks
  - $n^2/p$ elements / block
- Communicate at boundaries
  - $n/p$ per neighbor
  - Small if $n \gg p$



# Use ghost zones/nodes to buffer neighboring data.

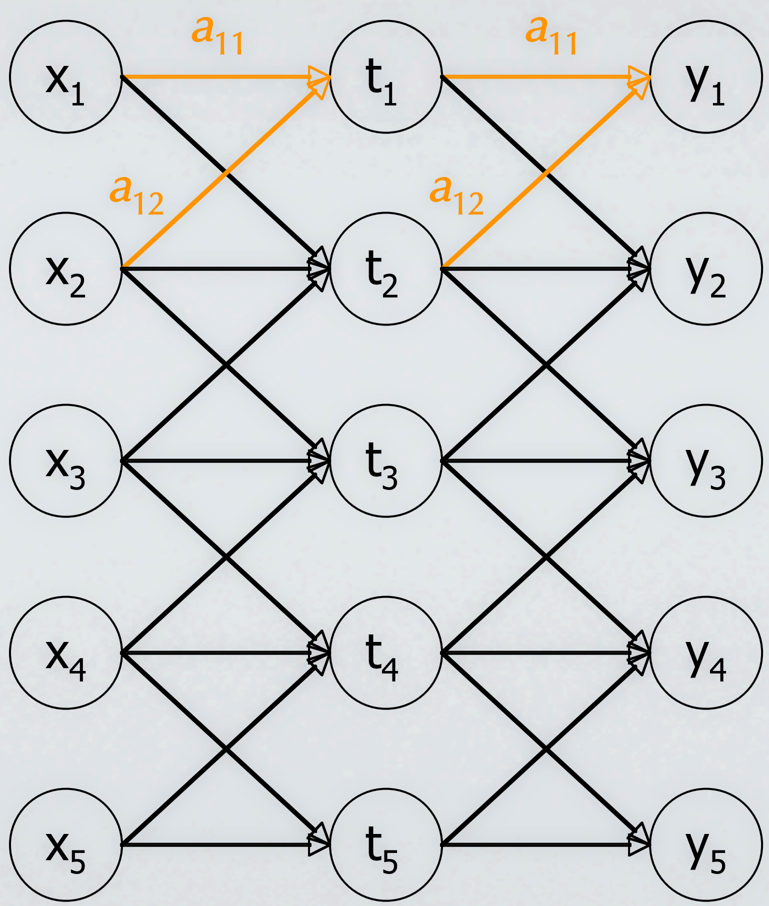
Block partition domain





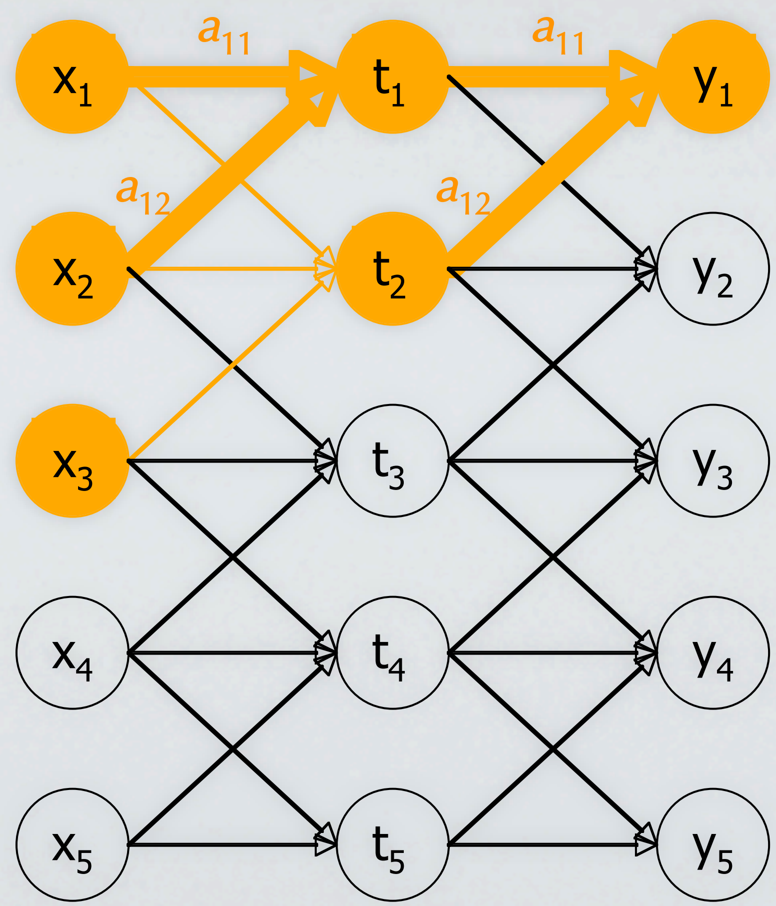
# Note: Each iteration is a matrix-vector multiply.

$$
\begin{aligned}
u_{i,j}^{t+1} &= \frac{1}{4}\left(u_{i-1,j}^{t} + u_{i+1,j}^{t} + u_{i,j-1}^{t} + u_{i,j+1}^{t} + h^2 f_{i,j}\right) \\
&\Longrightarrow \\
u^{t+1} &= \frac{1}{4}(T - I)u^{t} + f
\end{aligned}
$$



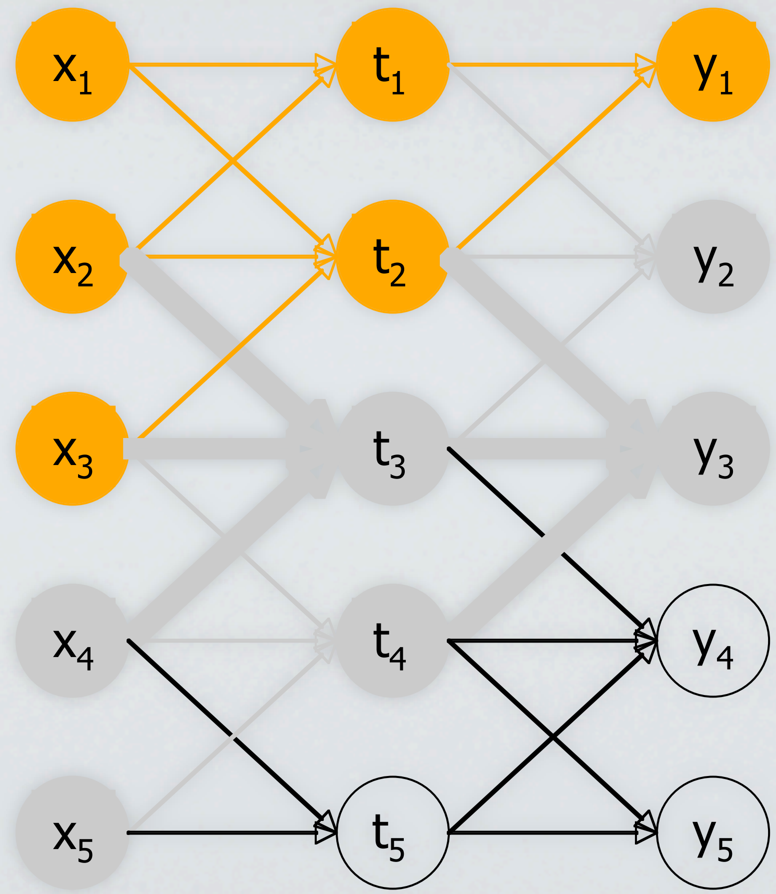

# What about locality?

Recall powers-kernel example from Lecture 1: $y = A^2 * x$




# What about locality?

Serial sparse tiling algorithm (Strout, *et al.*, 2001)




# What about locality?

Serial sparse tiling algorithm (Strout, *et al.*, 2001)

# Convergence of Jacobi's method

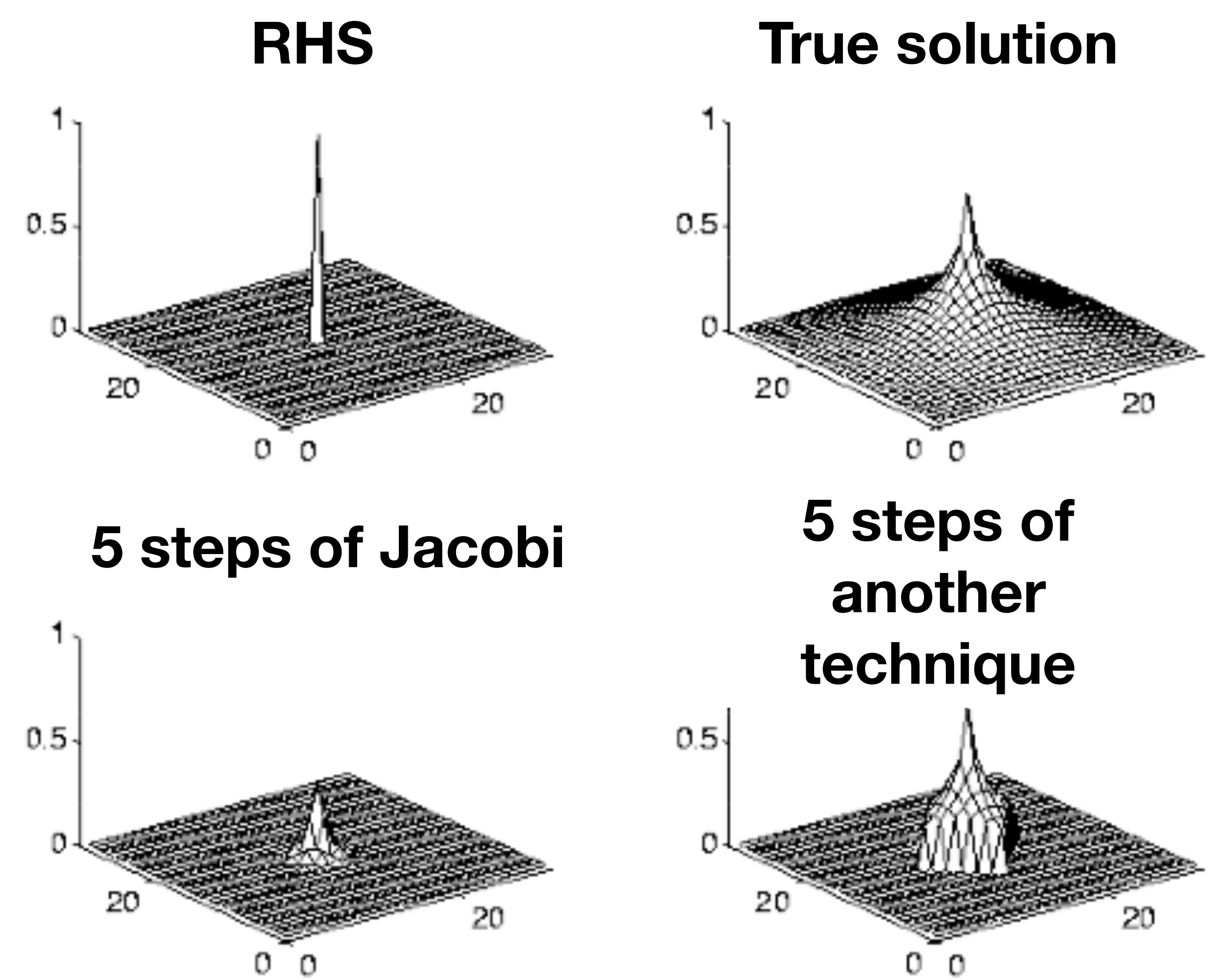
- Converges in *O(N=n²)* steps, so serial complexity is *O(N²)*.
- Define error at each step as:

$$\epsilon_t \triangleq \sqrt{\sum_{i,j} (u^t_{i,j} - u_{i,j})^2}$$

- For Jacobi, can show:

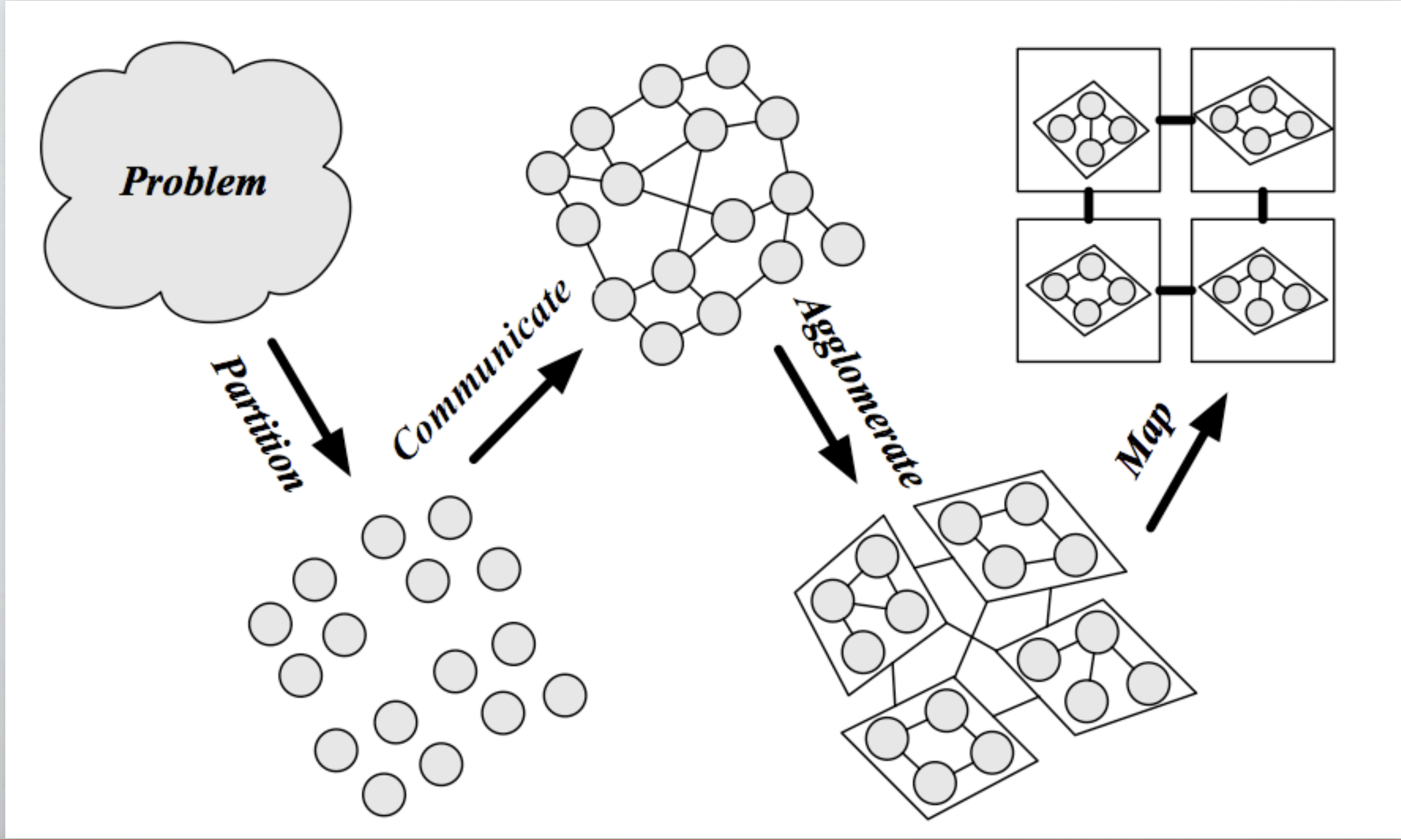
$$\epsilon_t \leq \left(\cos \frac{\pi}{n+1}\right)^t \epsilon_0 \quad \overset{n\to\infty}{\approx} \quad \left(1 - \frac{\pi^2}{4} \cdot \frac{t}{n^2}\right) \epsilon_0$$

# Numerical example illustrating slow convergence



“In conclusion…”

# Summary: Parallelization process



# Summary: Parallelization process

- Lots of opportunities to use “math” to solve interesting problems
- Four-step parallelization methodology
  - **Partition**: Identify fine-grained tasks
  - Determine **communication** pattern among tasks
  - **Agglomerate** fine-grained tasks into coarse-grained tasks to control communication requirements/overheads
  - **Map** tasks to processors