
Massively Parallel Computing with Cuda

- Grid Discretizations -

Hendrik Lensch
Robert Strzodka

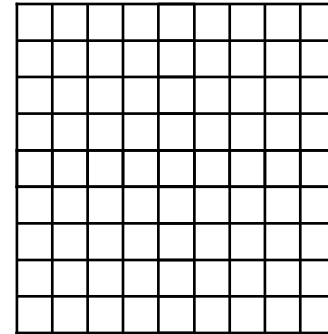
Today

- **Discrete Grids**
 - Equidistant, Tensor-product
 - Adaptive, Unstructured
 - PDE Motivation
 - Arrays, Hashes, Trees
 - Hybrid Data Layouts

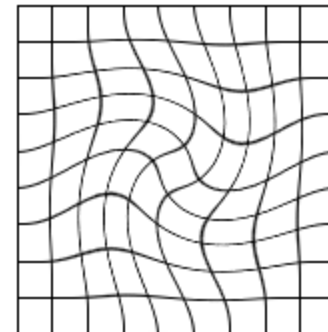
- **Project Proposals**

Discretization Grids

- **Equidistant grid**
 - topology: implicit
 - geometry: implicit
 - access: **direct**



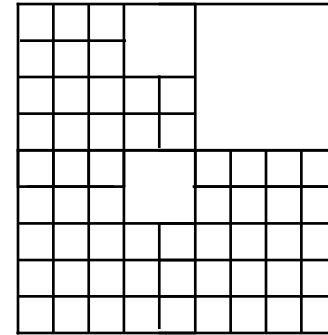
- **Generalized tensor-product grid**
 - topology: implicit
 - geometry: explicit
 - access: **direct**



Discretization Grids

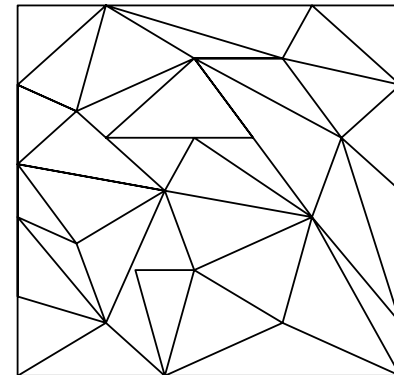
- Adaptive grid

- topology: **explicit**
- geometry: implicit/explicit
- access: hash, tree or page table



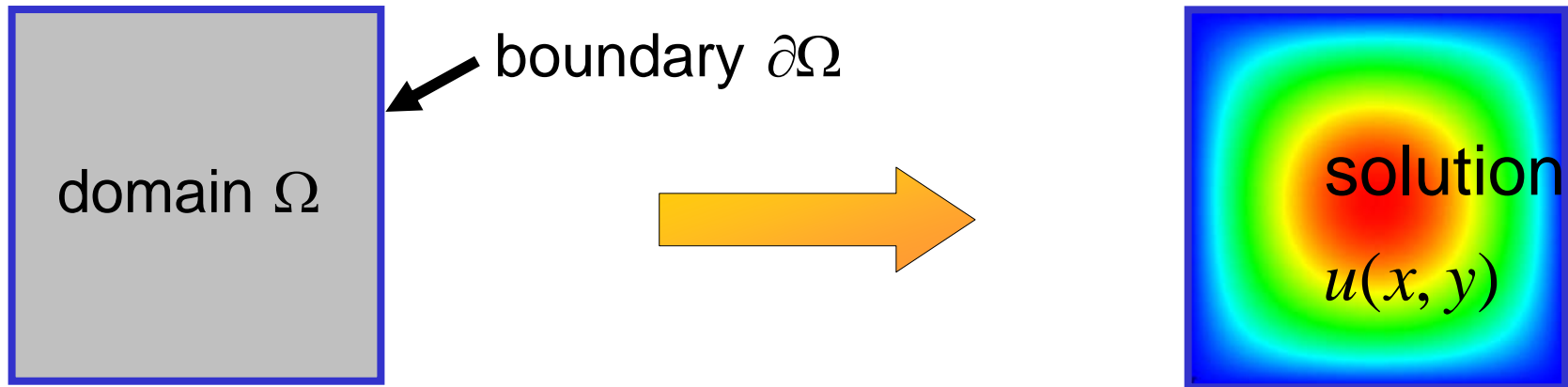
- Unstructured grid

- topology: **explicit**
- geometry: explicit
- access: index array



PDE Example

The Poisson Problem



Given function $b : \Omega \rightarrow \mathbf{R}^+$ find $u : \Omega \rightarrow \mathbf{R}^+$ such that

$$-\Delta u = b \quad \text{inside the domain } \Omega$$

$$u = 0 \quad \text{on the boundary } \partial\Omega$$

In 2D the Laplace operator is given as $\Delta u(x, y) := \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$

Discretization and Solvers

After discretization the Poisson Problem becomes a linear equation system

$$-\Delta u = b \qquad A\bar{u} = \bar{b}$$

For **small systems**, $Au=b$ is typically solved with a LU decomposition

$$PA = LU$$
$$L\bar{y} = P\bar{b}, \quad U\bar{u} = \bar{y}$$

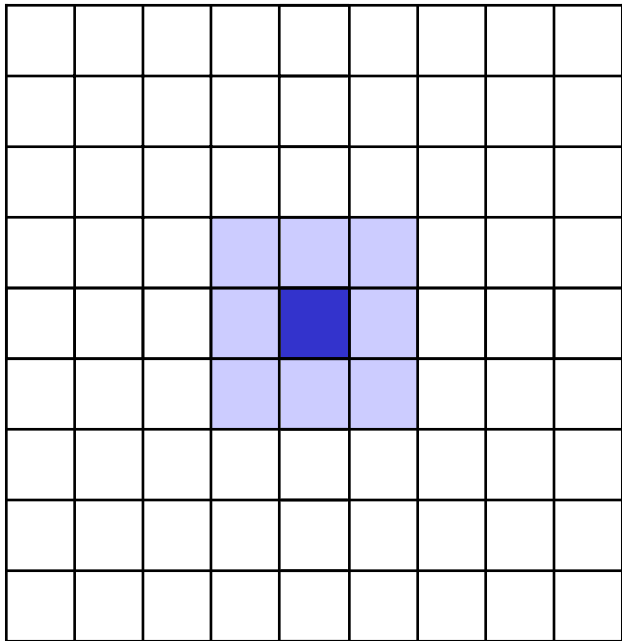
For **large systems**, $Au=b$ is typically solved with iterative schemes

$$\bar{u}^0 := \text{initial guess}$$
$$\bar{u}^{k+1} := \bar{u}^k + G(\bar{u}^k)$$

We obtain a convergent series: $(\bar{u}^k)_k = \bar{u}^0, \bar{u}^1, \bar{u}^2, \dots \xrightarrow{k \rightarrow \infty} \bar{u}^*$

Matrix Vector Product as Stencil Operation

Step n



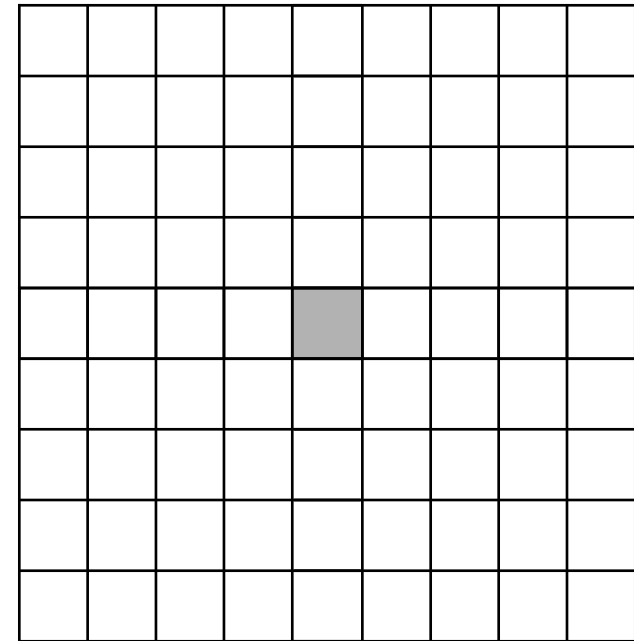
$$\left(\bar{v}_\beta^n \right)_{|\beta-\alpha| \leq C}$$

$$F_h \left(\left(\bar{v}_\beta^n \right)_{|\beta-\alpha| \leq C} \right)$$



$$\sum_{\beta: |\beta-\alpha| \leq C} A_{\alpha, \beta} \bar{v}_\beta^n$$

Step n+1



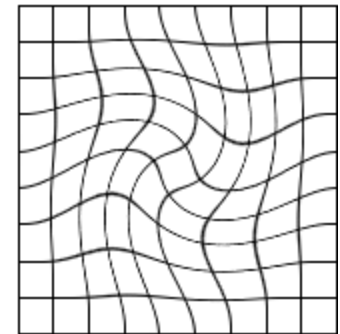
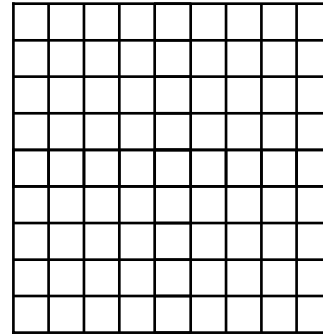
$$\bar{v}_\alpha^{n+1}$$

Parallel Grid Implementations

nD Arrays

- Generalized **tensor-product grid**

- topology: implicit
- geometry: implicit/explicit
- access: **direct**



- Pros

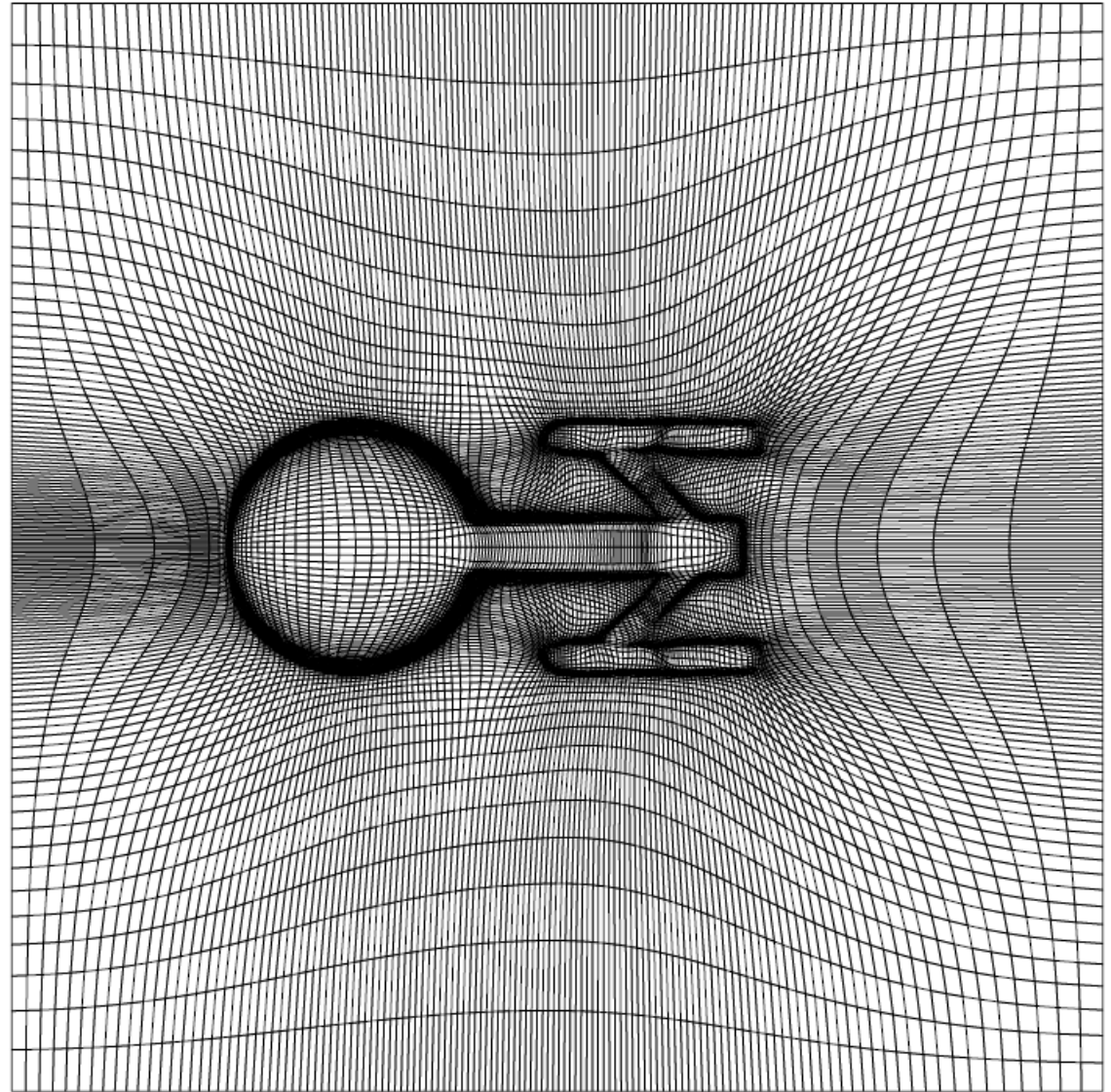
- **simple** implementation
- implicit topology saves **precious** global bandwidth
- texture (caches) particularly good for local stencil operations
- good SIMD utilization

- Cons

- topology constraints make object modeling more difficult
- element form may require a more powerful solver

Deformation Adaptivity

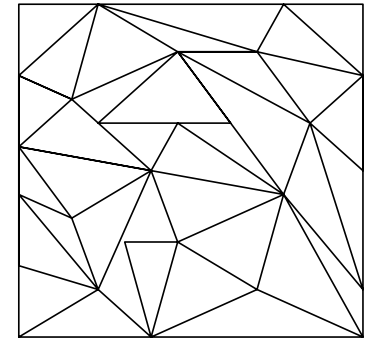
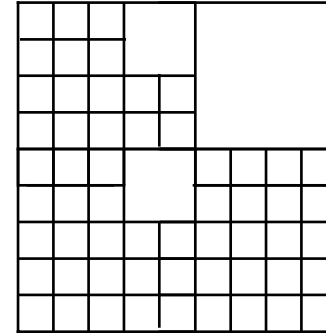
- This grid is a **tensor-product** !
- Easier to **accelerate in hardware** than resolution adaptive grids
- **Anisotropy level** determines optimal solver



nD Arrays

- Adaptive/Unstructured grid

- topology: **explicit**
- geometry: implicit/explicit
- access: indirect



- Pros

- general scheme for arbitrary node arrangements
- only one indirection for **local** data access
- clever node numberings preserve **some data locality**

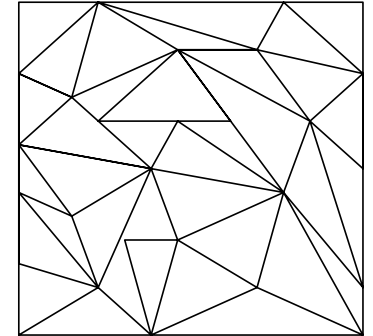
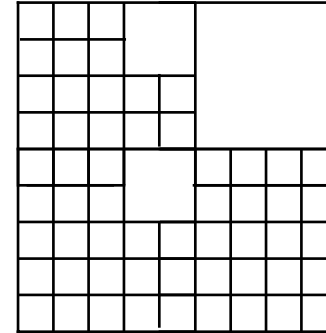
- Cons

- expensive encoding of topology/connectivity
- no global view of the structure
- difficult to handle dynamic changes in parallel

Hash

- Adaptive/Unstructured grid

- topology: **explicit**
- geometry: implicit/explicit
- access: indirect



- Pros

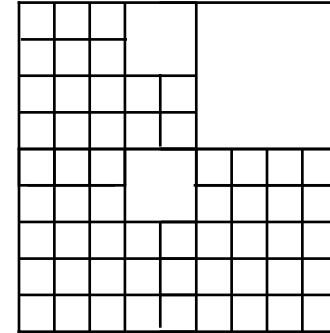
- general scheme for arbitrary node arrangements
- only one indirection for **arbitrary global** data access
- **perfect hashes** have no collisions, thus good SIMD use

- Cons

- expensive encoding of topology/connectivity
- hashes tend to produce fine-grained random data access
- perfect hash generation too expensive for dynamic changes

Tree

- **Adaptive grid**
 - topology: **explicit**
 - geometry: implicit/explicit
 - access: indirect

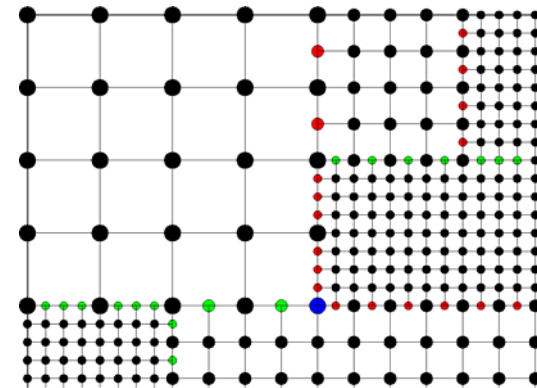


- **Pros**
 - allows refinement of arbitrary depths
 - compact encoding of **global** topology/connectivity
 - allows **dynamic changes** in parallel
- **Cons**
 - several memory indirection in data access
 - difficult to extract SIMD parallelism

Page Table = Arrays of Blocks

- Adaptive grid

- topology: **explicit**
- geometry: implicit/explicit
- access: indirect



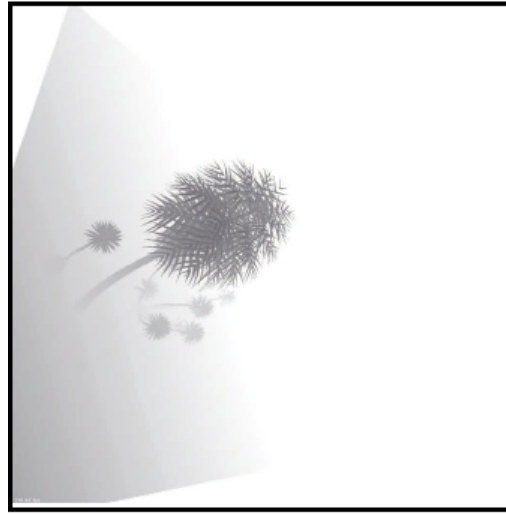
- Pros

- good SIMD utilization and data locality
- only one indirection for **arbitrary global** data access
- allows **dynamic changes** in parallel

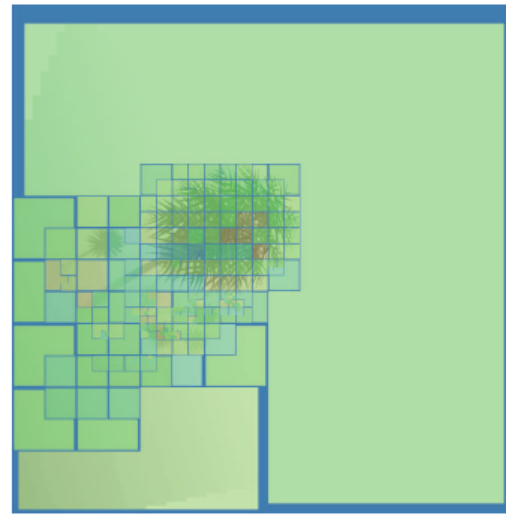
- Cons

- fixed finest resolution
- inefficient in worst case scenarios

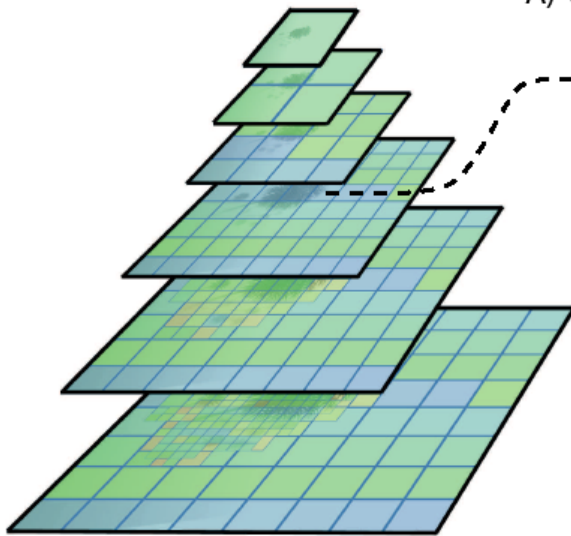
Page Tables in Glift



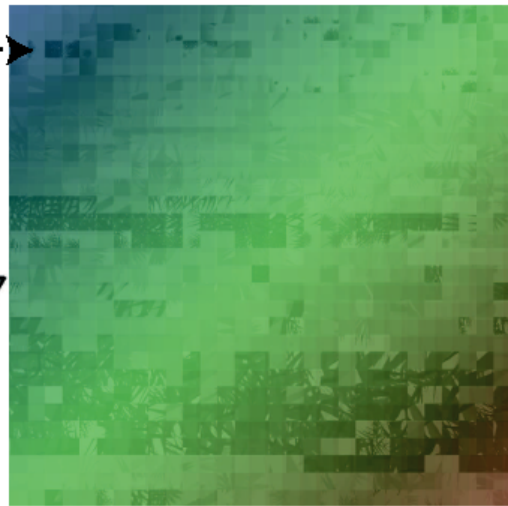
A) Virtual Domain



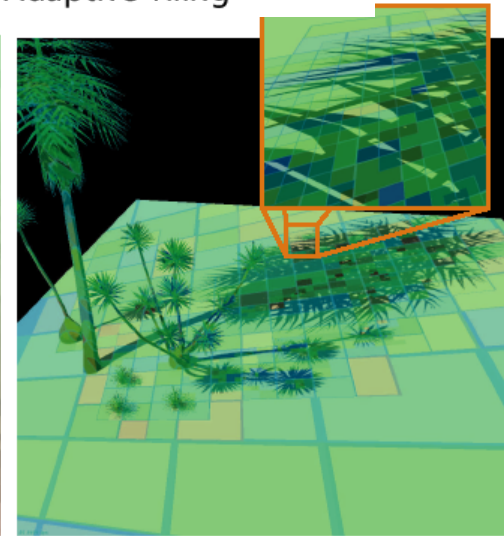
B) Adaptive Tiling



C) Page Table



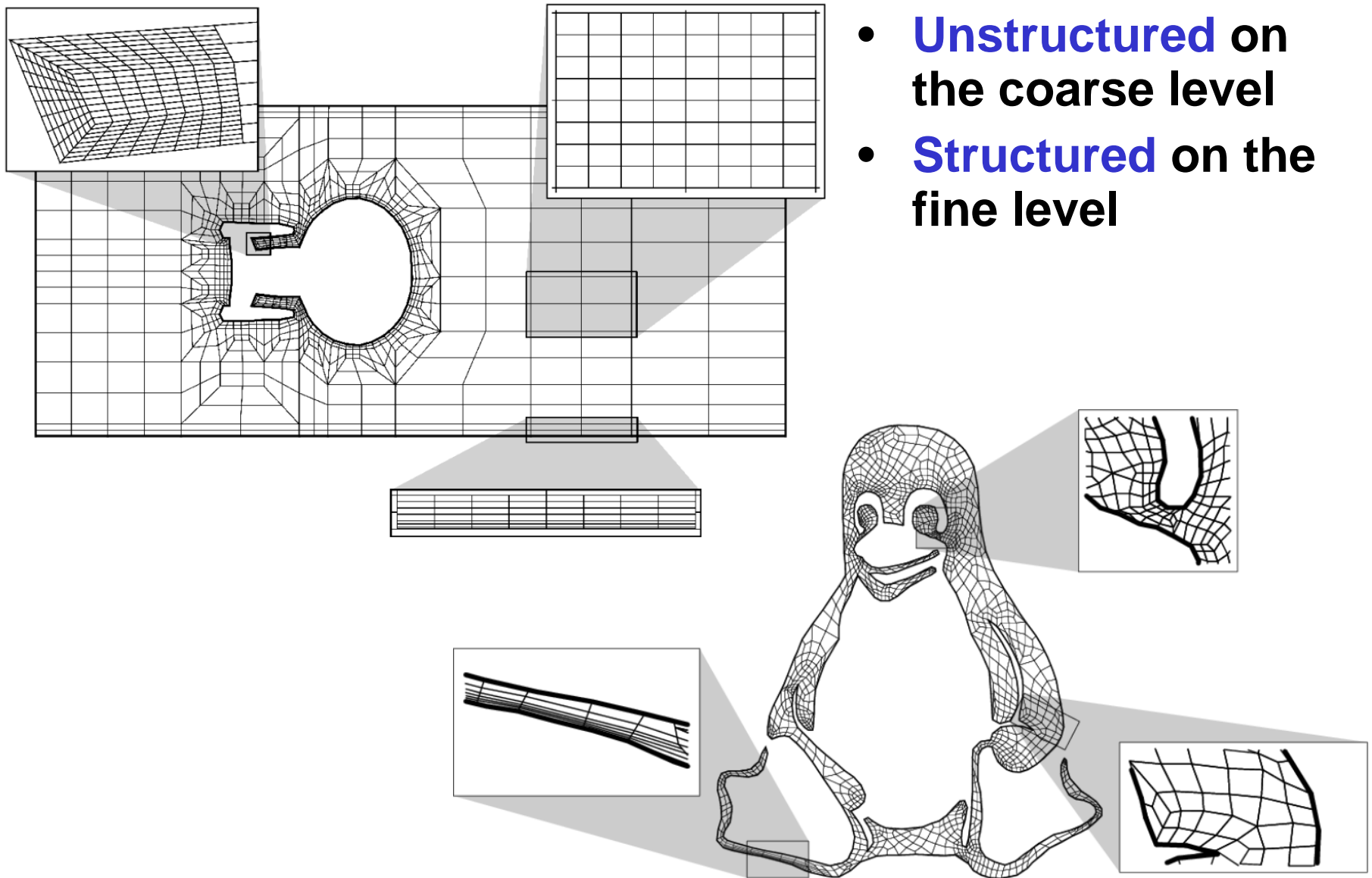
D) Physical Memory



E) Adaptive Shadow Map Rendering

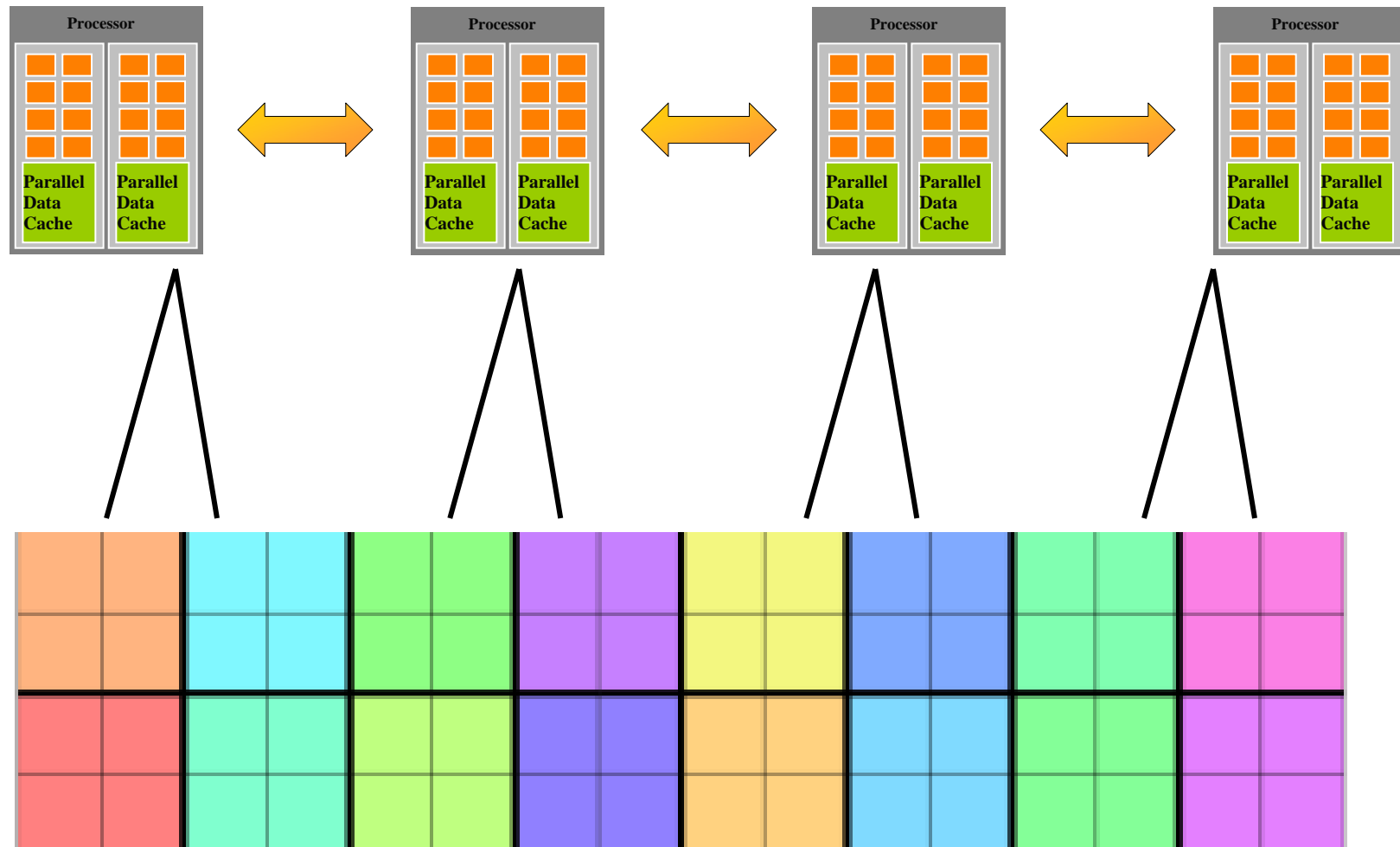
Hybrid Data Layouts

Generalized Tensor-Product Grids in FEAST



- **Unstructured** on the coarse level
- **Structured** on the fine level

Natural Domain Decomposition



Summary

- **Many choices for domain discretizations with grids**
 - Equidistant, tensor-product, adaptive, unstructured
 - Tradeoffs between ease of modeling and handling, required number of nodes for certain accuracy
 - Typically most important factor is fast access to neighbor elements

- **Many choices for implementation of grids**
 - Arrays, index structures, hashes, trees
 - Tradeoffs between data locality, required bandwidth, number of indirect memory accesses, SIMD parallelism, local vs. global data view, handling of dynamic updates

- **Hybrid Data Layouts**
 - Opportunities for maximizing benefits of different layouts