Dual Methods

- Set $\mathbf{x}^0 = (x_1^0, \ldots, x_n^0)$,
- For $k = 1, \ldots, K$
  - $x_i^{k+1} = \underset{y \in \mathbb{R}}{\arg\min}\, f(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, y, x_{i+1}^k, \ldots, x_n^k)$

source: wikipedia

Dual Methods

Reminders on Coordinate Descent

- Set $\mathbf{x}^0 = (x_1^0, \ldots, x_n^0)$,
- For $k = 1, \ldots, K$
  - $x_i^{k+1} = \arg\min_{y \in \mathbb{R}} f(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, y, x_{i+1}^k, \ldots, x_n^k)$

source: wikipedia

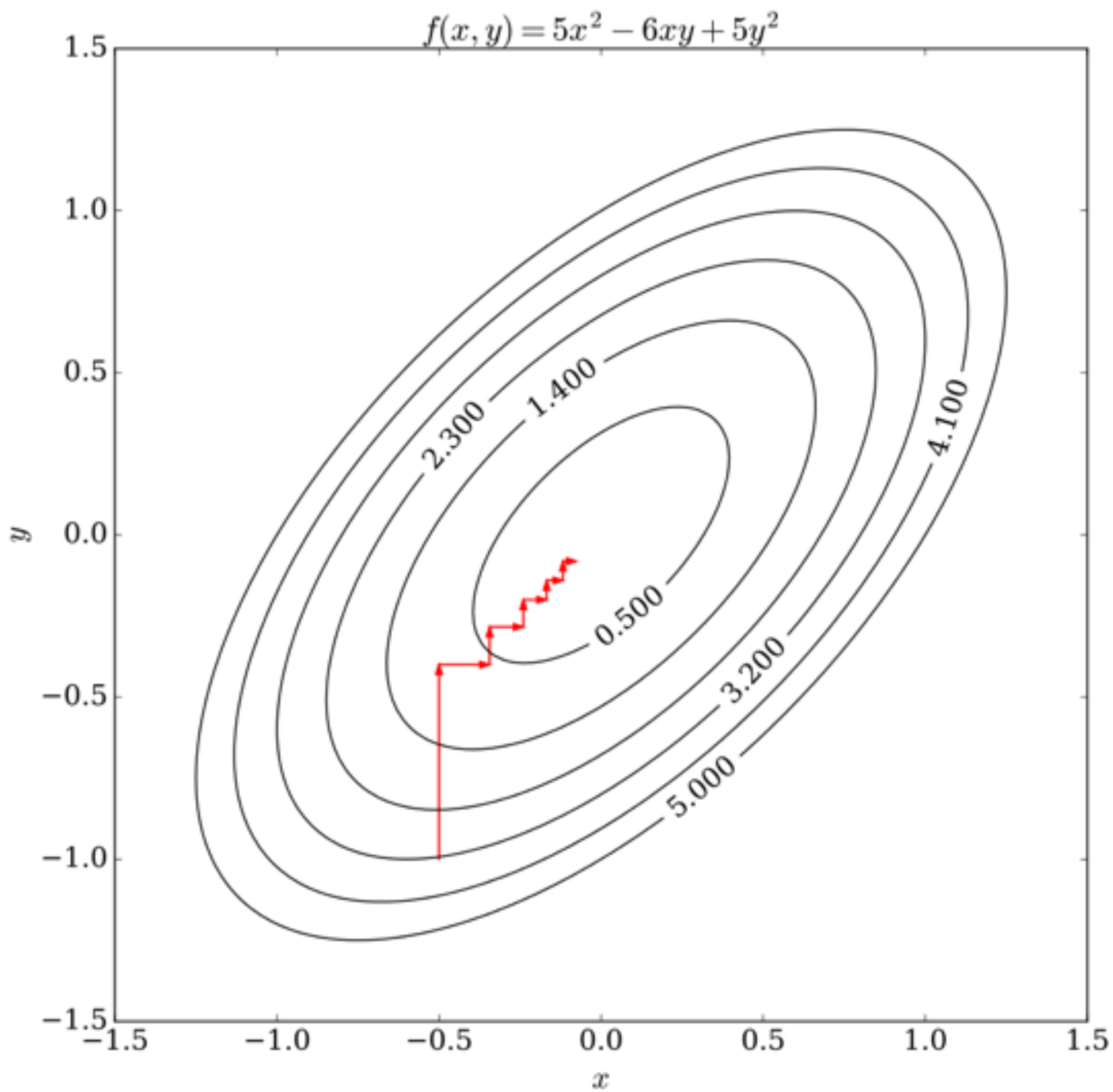
Dual Methods

Reminders on Coordinate Descent

source: wikipedia

Dual Methods

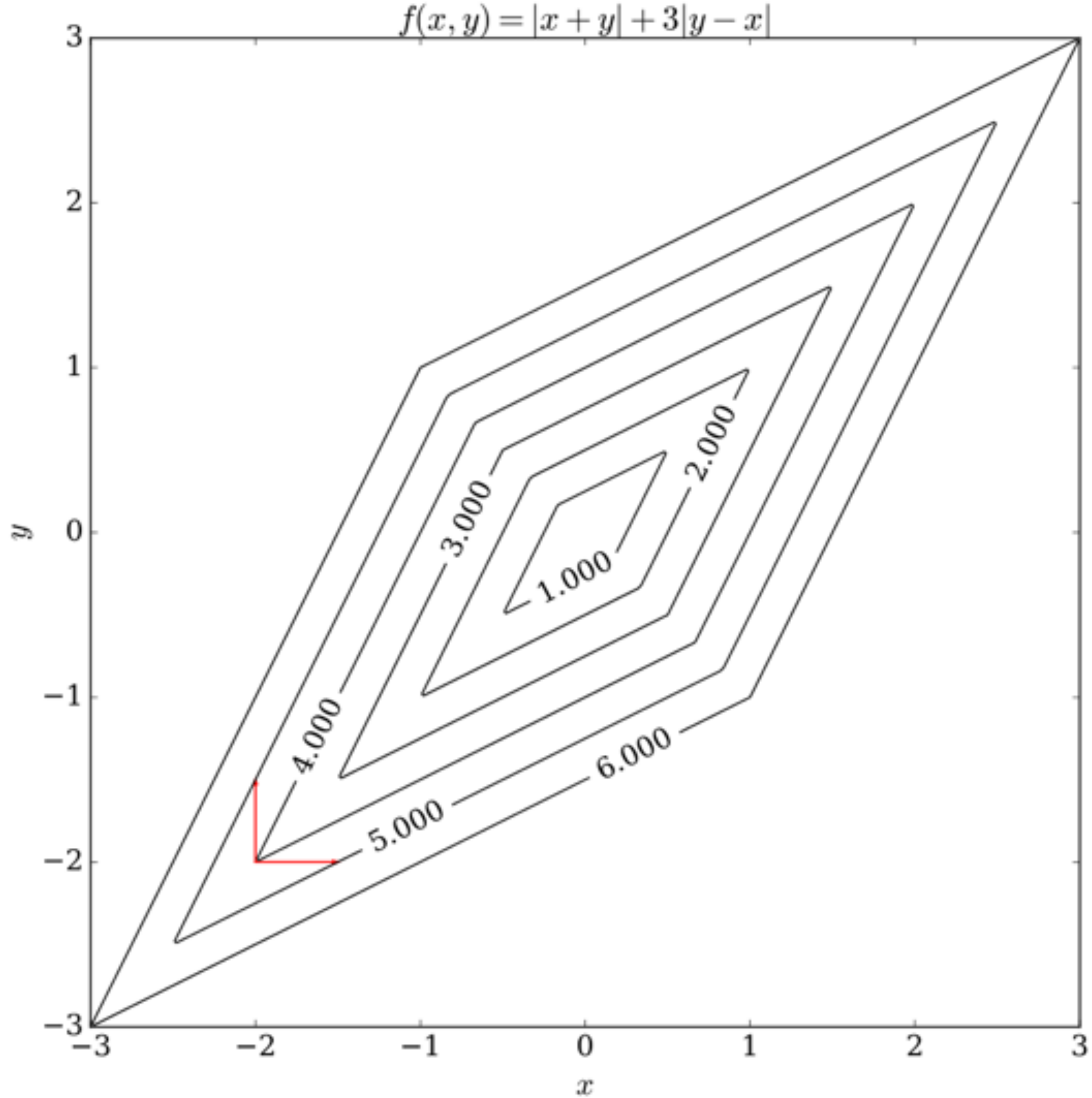
Reminders on Coordinate Descent



source: wikipedia

Dual Methods

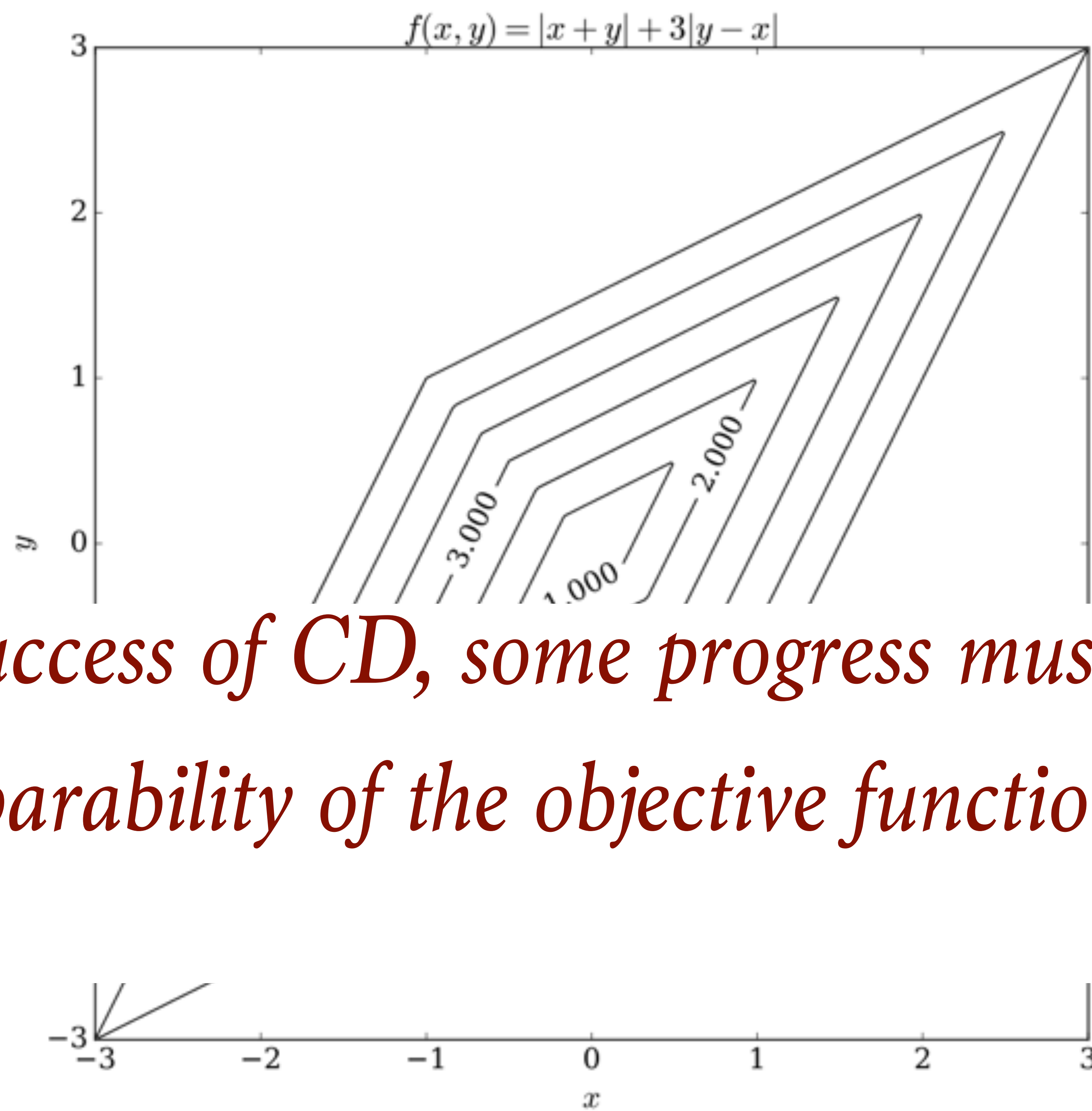
Reminders on Coordinate Descent



source: wikipedia

Dual Methods

Reminders on Coordinate Descent



To ensure success of CD, some progress must be guaranteed.

Separability of the objective function helps.

source: wikipedia

Dual Methods

- Set $\theta^0 = (\theta_1^0, \dots, \theta_p^0)$,
- For $k = 1, \dots, K$
  - Sample j.
  - Compute $g_j = \partial f(\theta)/\partial\theta_j$
  - $\theta_j \leftarrow \arg\min_{y\in\mathbb{R}} g_j y + \psi_j(y) + \frac{1}{2\eta_t}\|y - \theta_j\|^2$

source: wikipedia

Dual Methods

Coordinate Descent on Primal Problem

- Set $\theta^0 = (\theta_1^0, \ldots, \theta_p^0)$,
- For $k = 1, \ldots, K$
  - Sample j.
  - Compute $g_j = \partial f(\theta)/\partial \theta_j$
  - $\theta_j \leftarrow \underset{y \in \mathbb{R}}{\arg\min}\; g_j y + \psi_j(y) + \frac{1}{2\eta_t}\|y - \theta_j\|^2$

Regularizer must be separable.

source: wikipedia

Fenchel Duality Theorem

Theorem

Let $f : \mathbb{R}^p \to \bar{R}$ and $g : \mathbb{R}^q \to \bar{R}$ be closed convex, and $A \in \mathbb{R}^{q\times p}$ a linear map. Suppose that either condition (a) or (b) is satisfied. Then

$$\inf_{x\in\mathbb{R}^p} f(x) + g(Ax) = \sup_{y\in\mathbb{R}^q} -f^*(A^T y) - g^*(-y)$$

$$\boxed{\min_{\boldsymbol{\theta}\in\mathbb{R}^p} \frac{1}{n}\sum_{i=1}^{n} l_{\boldsymbol{\theta}}(z_i) + \psi(\boldsymbol{\theta})}$$

$l_{\boldsymbol{\theta}}(z_i) = h(y_i \times x_i^T \boldsymbol{\theta})$

$l(\rho) = \sum_i h(y_i \times x_i^T \boldsymbol{\theta})$

$$\boxed{\min_{\boldsymbol{\theta}\in\mathbb{R}^p} \frac{1}{n} l(X\boldsymbol{\theta}) + \psi(\boldsymbol{\theta})}$$

$$\boxed{\inf_{\boldsymbol{u}\in\mathbb{R}^{\boldsymbol{n}}} \frac{1}{n}\sum_{i} l_i^*(u_i) + \psi^*(-X^T u/n)}$$

if ψ differentiable,

$$\boxed{\boldsymbol{\theta}^* = \nabla\psi^*(-X^T \boldsymbol{u}^*/n)}$$

Losses

Name	Loss $\ell_i(z)$	Conjugate loss $\ell_i^*(u)$
Hinge	$\max\{0, 1 - y_i z\}$	$\ell_i^*(u) = \begin{cases} y_i u, & -1 \le y_i u \le 0, \\ +\infty, & \text{otherwise} \end{cases}$
Square hinge	$\max\{0, 1 - y_i z\}^2$	$\ell_i^*(u) = \begin{cases} y_i u + \frac{u^2}{4}, & y_i u \le 0, \\ +\infty, & \text{otherwise} \end{cases}$
Linear or l1	$\|y_i - z\|$	$\ell_i^*(u) = \begin{cases} y_i u, & -1 \le y_i u \le 1, \\ +\infty, & \text{otherwise} \end{cases}$
Square or l2	$(y_i - z)^2$	$\ell_i^*(u) = y_i u + \dfrac{u^2}{4}$
Insensitive l1	$\max\{0, \|y_i - z\| - \epsilon\}.$	
Logistic	$\log(1 + e^{-y_i z})$	$\ell_i^*(u) = \begin{cases} (1+u)\log(1+u) - u\log(-u), & -1 \le y_i u \le 0, \\ +\infty, & \text{otherwise} \end{cases}$

http://www.vlfeat.org/api/svm-sdca.html

SDCA

SDCA

Problem: $\inf_{\boldsymbol{u} \in \mathbb{R}^n} \frac{1}{n} \sum_i l_i^*(\boldsymbol{u}_i) + \psi^*(-X^T \boldsymbol{u}/n)$

- For $t = 1, 2, \ldots,$,
  - Pick $i \in \{1, \ldots, n\}$ uniformly at random.
  - Update dual $\boldsymbol{u_i}$ so that objective decreases.

Use primal-dual relationship to recover θ.

$$\boldsymbol{\theta^*} = \nabla \psi^*(-X^T \boldsymbol{u^*}/n)$$

SDCA

SDCA

Problem: $\inf_{\boldsymbol{u}\in\mathbb{R}^n} \frac{1}{n}\sum_i l_i^*(\boldsymbol{u}_i) + \psi^*(-X^T\boldsymbol{u})$

- For $t = 1, 2, \ldots,$,
  - Pick $i \in \{1, \ldots, n\}$ uniformly at random.
  - Update dual $\boldsymbol{u}_i$ so that objective increases:

$$\boldsymbol{u}_i^t \in \arg\min_{\boldsymbol{u}\in\mathbb{R}} \frac{1}{n} l_i^*(\boldsymbol{u}) + \psi^*\left(-\left(X_i^T\boldsymbol{u} + X_{-i}^T\boldsymbol{u}_{-i}\right)/n\right) + \frac{1}{2\eta}\|\boldsymbol{u} - \boldsymbol{u}_i^{t-1}\|^2$$

Use primal-dual relationship to recover θ.

$$\boldsymbol{\theta}^* = \nabla\psi^*(-X^T\boldsymbol{u}^*/n)$$

SDCA

SDCA

Problem: $\inf_{\boldsymbol{u} \in \mathbb{R}^n} \frac{1}{n} \sum_i l_i^*(\boldsymbol{u}_i) + \psi^*(-X^T \boldsymbol{u})$

For $t = 1, 2, \ldots,$

- Compute $\boldsymbol{\theta}^{\boldsymbol{t-1}} = \nabla \psi^*(-X^T \boldsymbol{u}^{t-1}/n)$
  - Pick $i \in \{1, \ldots, n\}$ uniformly at random.
  - Update dual $\boldsymbol{u}_i$ so that objective increases:

$$\boldsymbol{u}_i^t \in \arg\min_{\boldsymbol{u} \in \mathbb{R}} \frac{1}{n} l_i^*(\boldsymbol{u}) + \langle \boldsymbol{\theta}^{\boldsymbol{t-1}}, X_i \boldsymbol{u} \rangle + \frac{1}{2\eta} \|\boldsymbol{u} - \boldsymbol{u}_i^{t-1}\|^2$$

A Concrete Example: SDCA/SVM

$$E(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n}\ell_i(\langle \mathbf{w}, \mathbf{x}\rangle).$$

$$D(\boldsymbol{\alpha}) = -\frac{1}{2\lambda n^2}\boldsymbol{\alpha}^\top X^\top X\boldsymbol{\alpha} + \frac{1}{n}\sum_{i=1}^{n} -\ell_i^*(-\alpha_i)$$

$$\mathbf{w}(\boldsymbol{\alpha}) = \frac{1}{\lambda n}\sum_{i=1}^{n}\mathbf{x}_i\alpha_i = \frac{1}{\lambda n}X\boldsymbol{\alpha}.$$

http://www.vlfeat.org/api/svm-sdca.html

SDCA, SVM ascent

$$D(\boldsymbol{\alpha}_t + \mathbf{e}_q \Delta\alpha_q) = \text{const.} - \frac{1}{2\lambda n^2}\mathbf{x}_q^\top \mathbf{x}_q (\Delta\alpha_q)^2 - \frac{1}{n}\mathbf{x}_q^\top \frac{X\alpha_t}{\lambda n}\Delta\alpha_q - \frac{1}{n}\ell_q^*(-\alpha_q - \Delta\alpha_q)$$

$$\mathbf{w}_t = \frac{X\boldsymbol{\alpha}_t}{\lambda n}, \qquad \mathbf{w}_{t+1} = \mathbf{w}_t + \frac{1}{\lambda n}\mathbf{x}_q \mathbf{e}_q \Delta\alpha_q.$$

http://www.vlfeat.org/api/svm-sdca.html

SDCA, SVM ascent

$$D(\boldsymbol{\alpha}_t + \mathbf{e}_q \Delta\alpha_q) = \text{const.} - \frac{1}{2\lambda n^2}\mathbf{x}_q^\top \mathbf{x}_q (\Delta\alpha_q)^2 - \frac{1}{n}\mathbf{x}_q^\top \frac{X\alpha_t}{\lambda n}\Delta\alpha_q - \frac{1}{n}\ell_q^*(-\alpha_q - \Delta\alpha_q)$$

$$D(\boldsymbol{\alpha}_t + \mathbf{e}_q \Delta\alpha_q) \propto -\frac{A}{2}(\Delta\alpha_q)^2 - B\Delta\alpha_q - \ell_q^*(-\alpha_q - \Delta\alpha_q),$$

$$A = \frac{1}{\lambda n}\mathbf{x}_q^\top \mathbf{x}_q = \frac{1}{\lambda n}\|\mathbf{x}_q\|^2,$$

$$B = \mathbf{x}_q^\top \frac{X\boldsymbol{\alpha}_t}{\lambda n} = \mathbf{x}_q^\top \mathbf{w}_t.$$

$$\mathbf{w}_t = \frac{X\boldsymbol{\alpha}_t}{\lambda n}, \qquad \mathbf{w}_{t+1} = \mathbf{w}_t + \frac{1}{\lambda n}\mathbf{x}_q \mathbf{e}_q \Delta\alpha_q.$$

http://www.vlfeat.org/api/svm-sdca.html

SDCA, SVM ascent, hinge

$$\ell_q^*(u) = \begin{cases} y_q u, & -1 \leq y_q u \leq 0, \\ +\infty, & \text{otherwise.} \end{cases}$$

Setting derivative to 0

$$\Delta\alpha_q = y_q \max\{0, \min\{1, y_q(\widetilde{\Delta\alpha_q} + \alpha_q)\}\} - \alpha_q.$$

http://www.vlfeat.org/api/svm-sdca.html

Distributed Optimization

Primal Methods

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^{\boldsymbol{p}}} \frac{1}{n} \sum_{i=1}^{n} l_i(\boldsymbol{\theta})$$

$\boxed{\text{We want to approximate } \nabla \frac{1}{n} \sum_{i=1}^{n} l_i(\boldsymbol{\theta})}$

Primal Methods

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^{\boldsymbol{p}}} \frac{1}{n} \sum_{i=1}^{n} l_i(\boldsymbol{\theta})$$

We want to approximate $\nabla \frac{1}{n} \sum_{i=1}^{n} l_i(\boldsymbol{\theta})$

$$\mathbb{E}_{i \sim \text{unif}\{1,\ldots,\text{n}\}}[\nabla l_i(\boldsymbol{\theta})] = \frac{1}{n} \sum_{i} \nabla l_i(\boldsymbol{\theta}) = \nabla \mathcal{L}(\boldsymbol{\theta})$$

Primal Methods

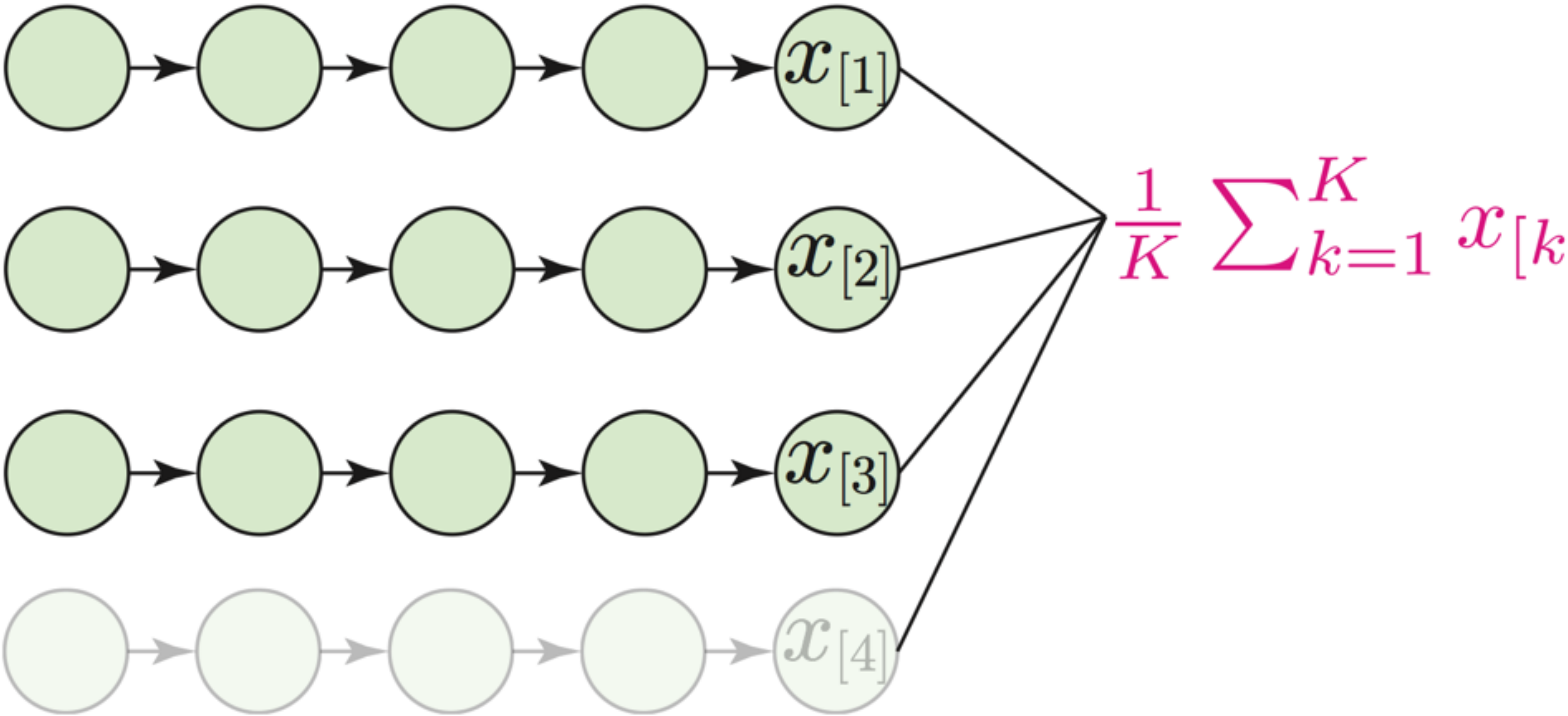
$$\min_{\boldsymbol{\theta} \in \mathbb{R}^{\boldsymbol{p}}} \frac{1}{n} \sum_{i=1}^{n} l_i(\boldsymbol{\theta})$$

$n \approx \infty$

$\boxed{\text{We want to approximate } \nabla \frac{1}{n} \sum_{i=1}^{n} l_i(\boldsymbol{\theta})}$

$$\mathbb{E}_{i \sim \mathrm{unif}\{1,\ldots,\mathrm{n}\}}[\nabla l_i(\boldsymbol{\theta})] = \frac{1}{n} \sum_{i} \nabla l_i(\boldsymbol{\theta}) = \nabla \mathcal{L}(\boldsymbol{\theta})$$

Distributed Computations



$$\hat{x}_K = \frac{1}{K}\sum_{k=1}^{K} x_{[k]}.$$

source: T. Suzuki

Distributed Computations

Theorem ((Zhang et al., 2013))

With an appropriate step size,

$$\mathrm{E}[\|\hat{x}_K - x^*\|^2] \leq C\left(\frac{G^2}{KT\lambda^2} + \frac{1}{T^{3/2}}\right).$$

$$\lambda \in (0, 1/\sqrt{T}) \quad \mathrm{E}[\|\hat{x}_K - x^*\|^2] \geq \frac{C}{\lambda^2 T}.$$

source: T. Suzuki

Distributed Computations

Assumptions

Loss is sufficiently **smooth**, **strongly** *convex, each node runs T iterations of SGD.*

Theorem ((Zhang et al., 2013))

With an appropriate step size,

$$\mathrm{E}[\|\hat{x}_K - x^*\|^2] \leq C\left(\frac{G^2}{KT\lambda^2} + \frac{1}{T^{3/2}}\right).$$

K improves performance **linearly.**

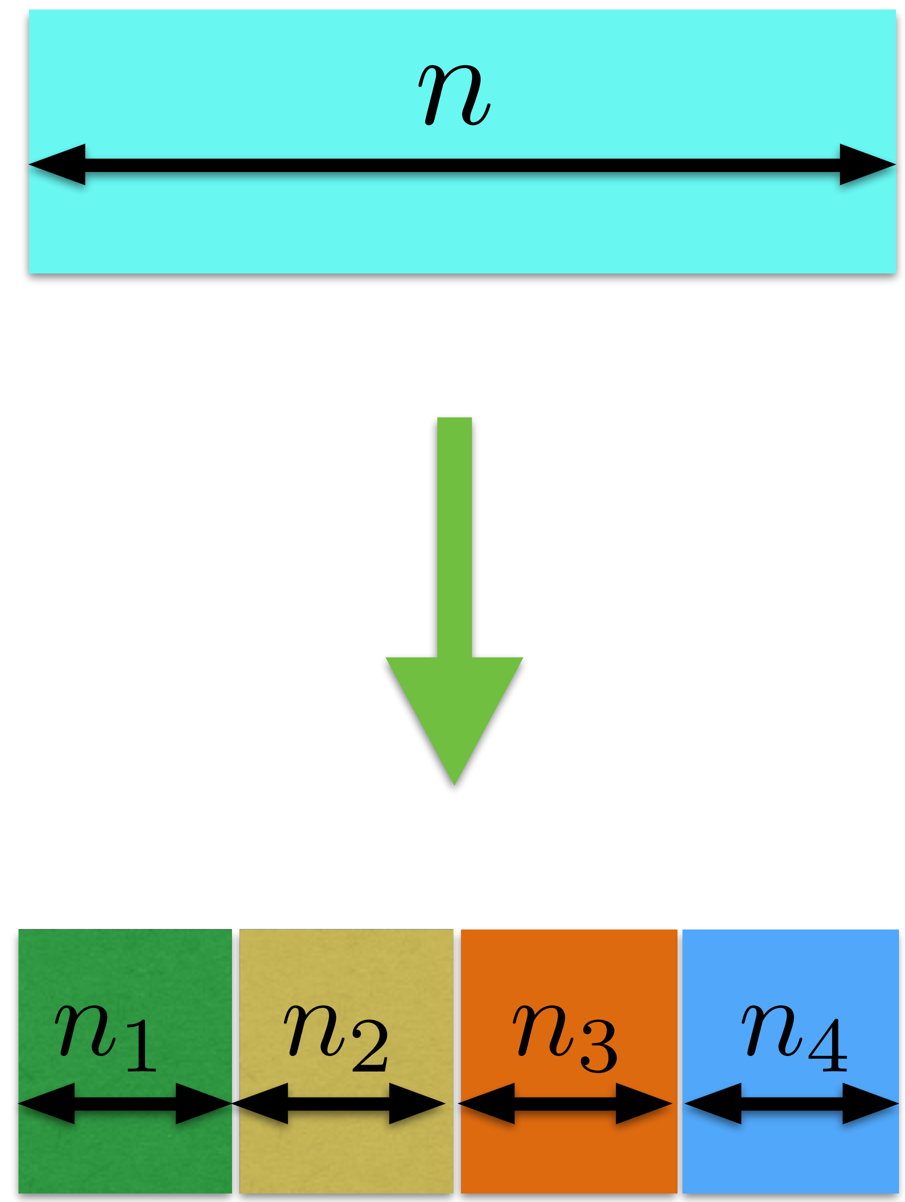
One can show however that if $\lambda \in (0, 1/\sqrt{T})$ $\mathrm{E}[\|\hat{x}_K - x^*\|^2] \geq \frac{C}{\lambda^2 T}$.

source: T. Suzuki

Distributed Primal Methods

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{\boldsymbol{p}}} \frac{1}{n}\sum_{i=1}^{n} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{\boldsymbol{p}}} \sum_{j=1}^{N} \left(\frac{1}{n_j}\sum_{i\in I_j} l_i(\boldsymbol{\theta})\right)$$

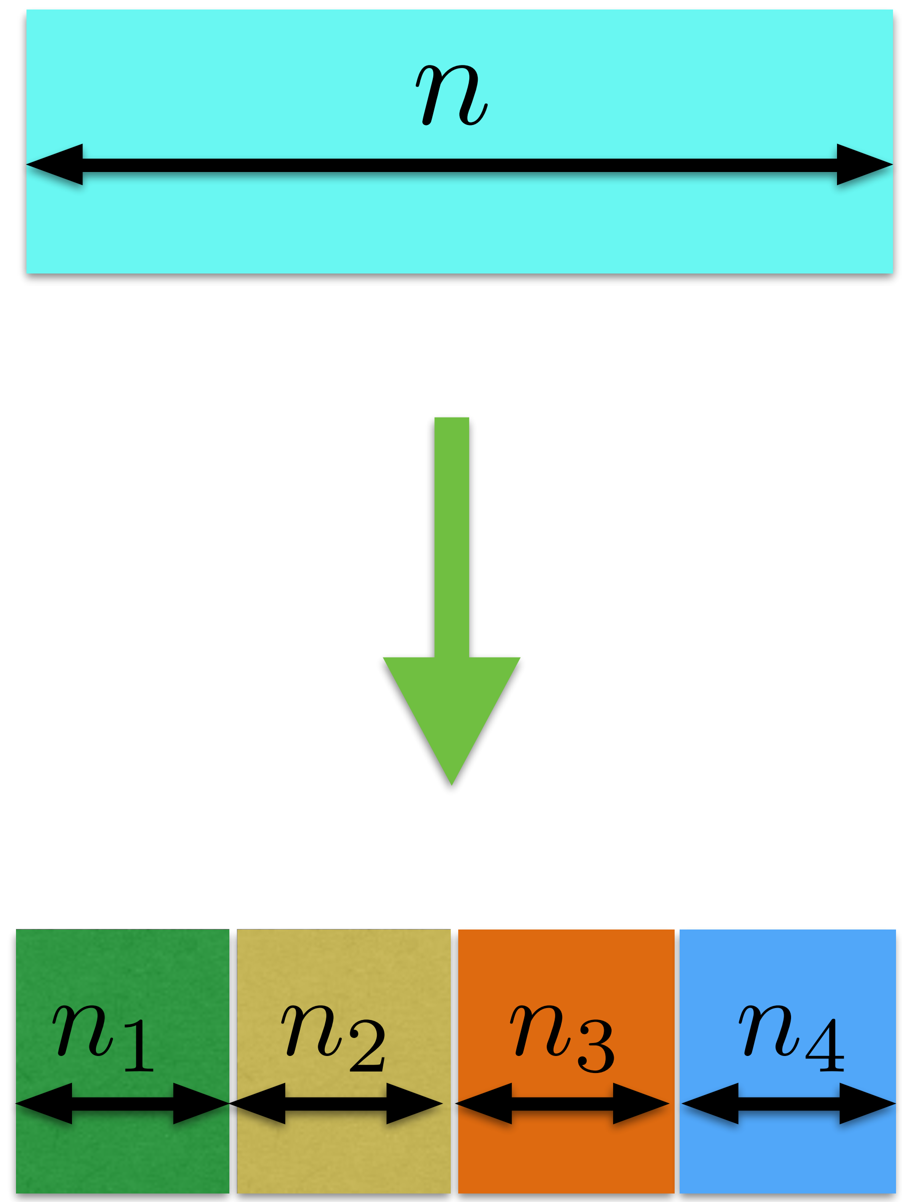


Distributed Primal Methods

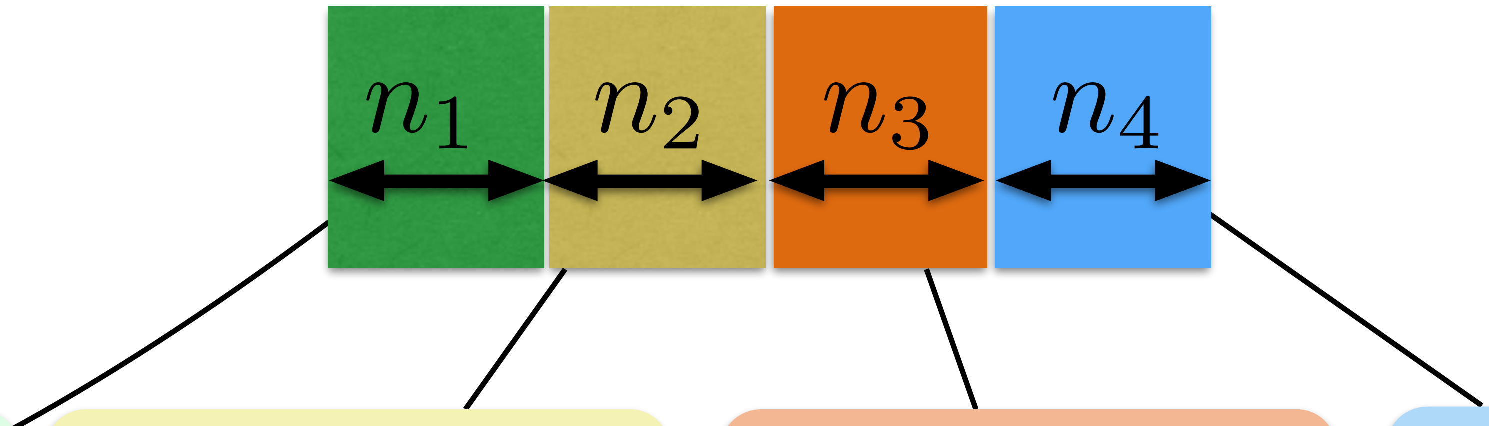
$$\min_{\boldsymbol{\theta} \in \mathbb{R}^{\boldsymbol{p}}} \frac{1}{n} \sum_{i=1}^{n} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^{\boldsymbol{p}}} \sum_{j=1}^{N} \left(\frac{1}{n_j} \sum_{i \in I_j} l_i(\boldsymbol{\theta}) \right)$$

$n \approx \infty$



Distributed Primal Methods



$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{\boldsymbol{p}}} \frac{1}{n_{\mathbf{1}}} \sum_{i\in I_{\mathbf{1}}} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{\boldsymbol{p}}} \frac{1}{n_{\mathbf{2}}} \sum_{i\in I_{\mathbf{2}}} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{\boldsymbol{p}}} \frac{1}{n_{\mathbf{3}}} \sum_{i\in I_{\mathbf{3}}} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{\boldsymbol{p}}} \frac{1}{n_{\mathbf{4}}} \sum_{i\in I_{\mathbf{4}}} l_i(\boldsymbol{\theta})$$

Distributed Primal Methods

100% Parallel

n_1 n_2 n_3 n_4

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n_1} \sum_{i \in I_1} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n_2} \sum_{i \in I_2} l_i(\boldsymbol{\theta})$$

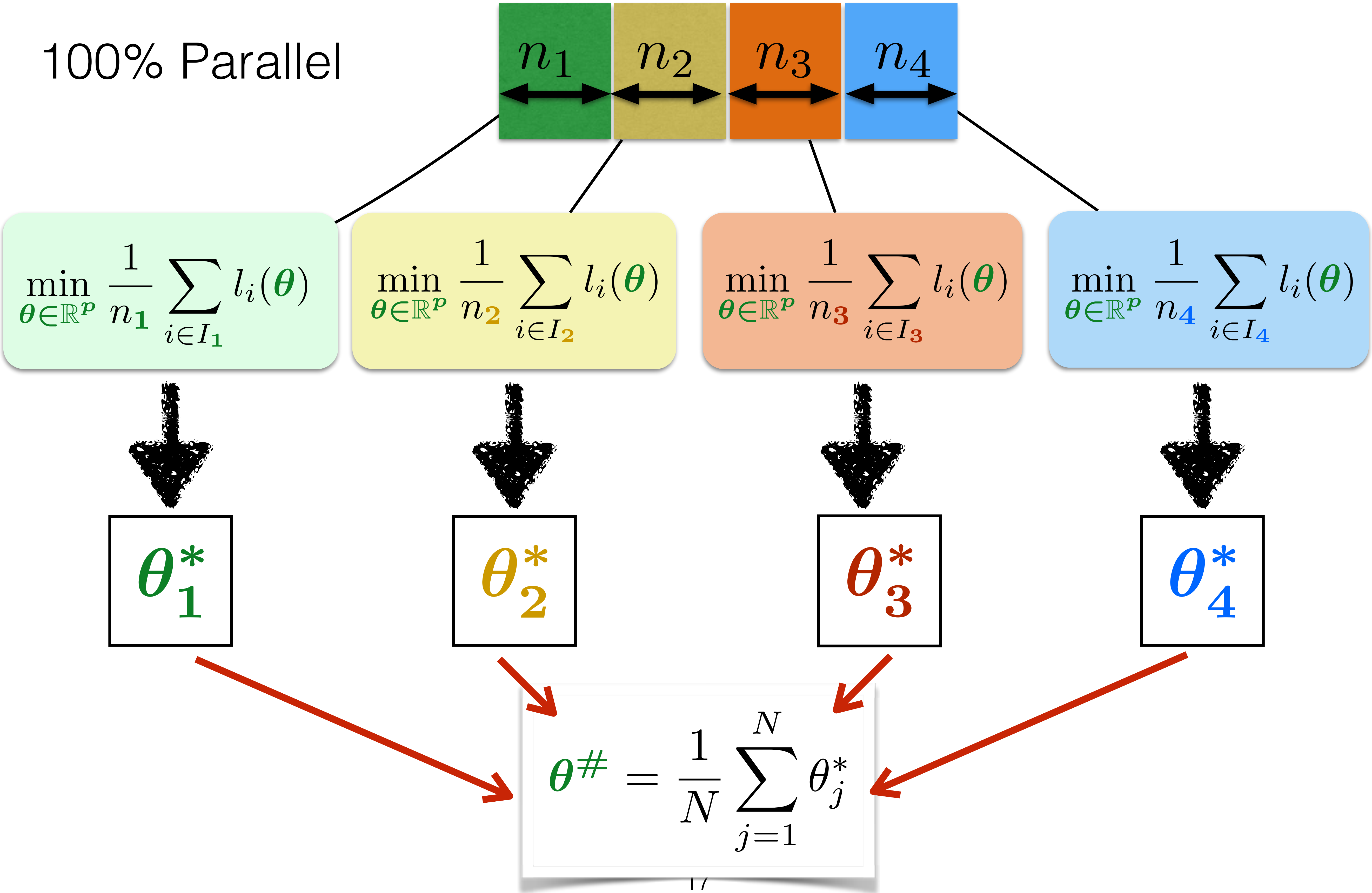
$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n_3} \sum_{i \in I_3} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n_4} \sum_{i \in I_4} l_i(\boldsymbol{\theta})$$

$\boxed{\boldsymbol{\theta}_1^*}$ $\boxed{\boldsymbol{\theta}_2^*}$ $\boxed{\boldsymbol{\theta}_3^*}$ $\boxed{\boldsymbol{\theta}_4^*}$

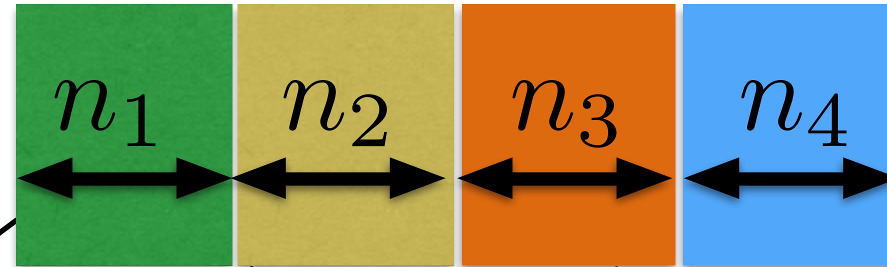
Distributed Primal Methods

100% Parallel



Distributed Primal Methods

Only gradient parallel



$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n_{\mathbf{1}}} \sum_{i \in I_{\mathbf{1}}} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n_{\mathbf{2}}} \sum_{i \in I_{\mathbf{2}}} l_i(\boldsymbol{\theta})$$

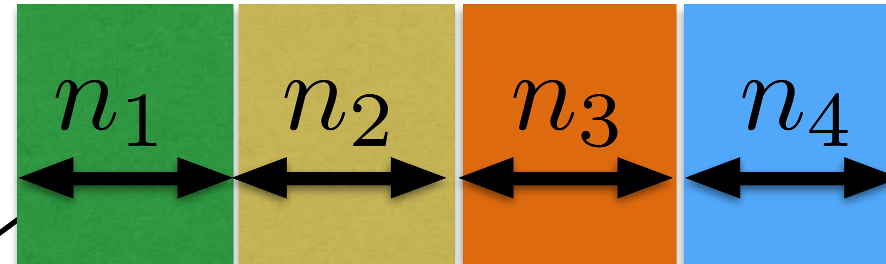
$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n_{\mathbf{3}}} \sum_{i \in I_{\mathbf{3}}} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n_{\mathbf{4}}} \sum_{i \in I_{\mathbf{4}}} l_i(\boldsymbol{\theta})$$

$$\boldsymbol{\nabla}_{\mathbf{1}}(\boldsymbol{\theta}_{\mathbf{0}}) \quad \boldsymbol{\nabla}_{\mathbf{2}}(\boldsymbol{\theta}_{\mathbf{0}}) \quad \boldsymbol{\nabla}_{\mathbf{3}}(\boldsymbol{\theta}_{\mathbf{0}}) \quad \boldsymbol{\nabla}_{\mathbf{4}}(\boldsymbol{\theta}_{\mathbf{0}})$$

Distributed Primal Methods

Only gradient parallel



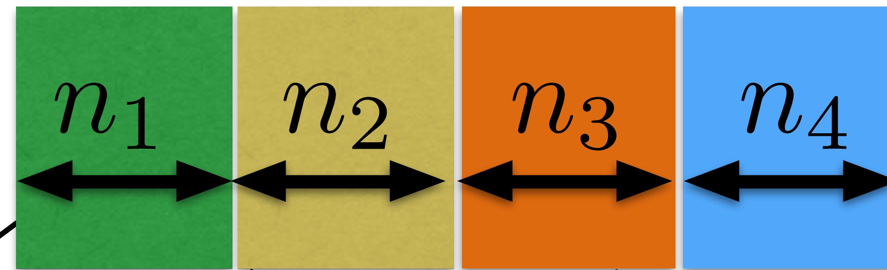
$$\min_{\theta \in \mathbb{R}^p} \frac{1}{n_1} \sum_{i \in I_1} l_i(\theta) \qquad \min_{\theta \in \mathbb{R}^p} \frac{1}{n_2} \sum_{i \in I_2} l_i(\theta) \qquad \min_{\theta \in \mathbb{R}^p} \frac{1}{n_3} \sum_{i \in I_3} l_i(\theta) \qquad \min_{\theta \in \mathbb{R}^p} \frac{1}{n_4} \sum_{i \in I_4} l_i(\theta)$$

$$\nabla_1(\theta_0) \qquad \nabla_2(\theta_0) \qquad \nabla_3(\theta_0) \qquad \nabla_4(\theta_0)$$

$$\nabla(\theta_0) = \frac{n_1 \nabla_1 + n_2 \nabla_2 + n_3 \nabla_3 + n_4 \nabla_4}{n}$$

Distributed Primal Methods

Only gradient parallel



$$\min_{\boldsymbol{\theta}\in\mathbb{R}^p} \frac{1}{n_1} \sum_{i\in I_1} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^p} \frac{1}{n_2} \sum_{i\in I_2} l_i(\boldsymbol{\theta})$$

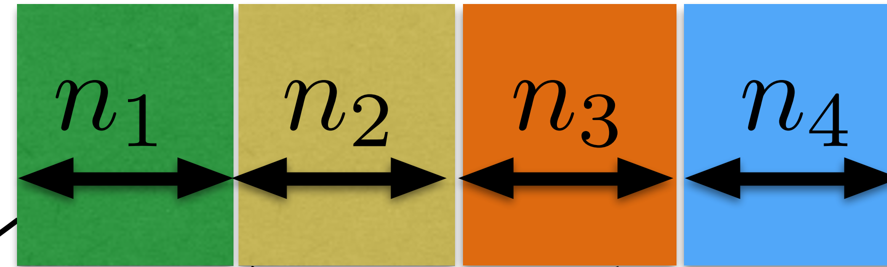
$$\min_{\boldsymbol{\theta}\in\mathbb{R}^p} \frac{1}{n_3} \sum_{i\in I_3} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^p} \frac{1}{n_4} \sum_{i\in I_4} l_i(\boldsymbol{\theta})$$

$$\boldsymbol{\theta}_1 = \nabla(\boldsymbol{\theta}_0) - \rho\nabla(\boldsymbol{\theta}_0)$$

Distributed Primal Methods

Only gradient parallel



$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{p}} \frac{1}{n_{\mathbf{1}}} \sum_{i\in I_{\mathbf{1}}} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{p}} \frac{1}{n_{\mathbf{2}}} \sum_{i\in I_{\mathbf{2}}} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{p}} \frac{1}{n_{\mathbf{3}}} \sum_{i\in I_{\mathbf{3}}} l_i(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{p}} \frac{1}{n_{\mathbf{4}}} \sum_{i\in I_{\mathbf{4}}} l_i(\boldsymbol{\theta})$$

$$\boldsymbol{\theta}_{\mathbf{1}} = \nabla(\boldsymbol{\theta}_{\mathbf{0}}) - \rho\nabla(\boldsymbol{\theta}_{\mathbf{0}})$$

Communication cost!!!

How can we incorporate regularizer?

ADMM & Splitting Methods

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{\boldsymbol{p}}} \sum_{j=1}^{N} \left(\frac{1}{n_j} \sum_{i\in I_j} l_i(\boldsymbol{\theta}) \right) = \min_{\boldsymbol{\theta}\in\mathbb{R}^{\boldsymbol{p}}} \sum_{j=1}^{N} f_j(\boldsymbol{\theta})$$

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{\boldsymbol{p}}} \sum_{j=1}^{N} f_j(\boldsymbol{\theta}) = \min_{\substack{\boldsymbol{\theta_1},\ldots,\boldsymbol{\theta_N}\in\mathbb{R}^{\boldsymbol{p}} \\ \boldsymbol{\theta_1}=\boldsymbol{\theta_2}=\cdots=\boldsymbol{\theta_N}}} \sum_{j=1}^{N} f_j(\boldsymbol{\theta_j})$$

ADMM & Splitting Methods

$$\boxed{\min_{\substack{\boldsymbol{\theta}_{\mathbf{1}},\ldots,\boldsymbol{\theta}_{\boldsymbol{N}} \in \mathbb{R}^{\boldsymbol{p}} \\ \boldsymbol{\rho}=\boldsymbol{\theta}_{\mathbf{1}}=\boldsymbol{\theta}_{\mathbf{2}}=\cdots=\boldsymbol{\theta}_{\boldsymbol{N}}}} \sum_{j=1}^{N} f_j(\boldsymbol{\theta}_{\boldsymbol{j}}) + \boldsymbol{\psi}(\boldsymbol{\rho})}$$

ADMM & Splitting Methods

The generic splitting problem we will address:

$$\boxed{\min_{\substack{\boldsymbol{\theta}_{\mathbf{1}},\ldots,\boldsymbol{\theta}_{\boldsymbol{N}} \in \mathbb{R}^{\boldsymbol{p}} \\ \boldsymbol{\rho}=\boldsymbol{\theta}_{\mathbf{1}}=\boldsymbol{\theta}_{\mathbf{2}}=\cdots=\boldsymbol{\theta}_{\boldsymbol{N}}}} \sum_{j=1}^{N} f_j(\boldsymbol{\theta}_{\boldsymbol{j}}) + \boldsymbol{\psi}(\boldsymbol{\rho})}$$

ADMM & Splitting Methods

$$\boxed{\min_{\substack{\boldsymbol{\theta}_{\mathbf{1}},\ldots,\boldsymbol{\theta}_{\boldsymbol{N}} \in \mathbb{R}^{\boldsymbol{p}} \\ \boldsymbol{\rho}=\boldsymbol{\theta}_{\mathbf{1}}=\boldsymbol{\theta}_{\mathbf{2}}=\cdots=\boldsymbol{\theta}_{\boldsymbol{N}}}} \sum_{j=1}^{N} f_j(\boldsymbol{\theta}_{\boldsymbol{j}}) + \boldsymbol{\psi}(\boldsymbol{\rho})}$$

ADMM & Splitting Methods

repeat for $t = 0, \ldots, T$

$$\boldsymbol{\theta}_{\mathbf{1}}^{t+1} = \underset{\theta}{\operatorname{argmin}}\, f_{\mathbf{1}}(\theta) + \frac{\tau}{2}\|\theta - \boldsymbol{\rho}^t + \boldsymbol{u}_{\mathbf{1}}^t\|^2$$

$$\vdots \quad \vdots$$

$$\boldsymbol{\theta}_{\boldsymbol{N}}^{t+1} = \underset{\theta}{\operatorname{argmin}}\, f_{\boldsymbol{N}}(\theta) + \frac{\tau}{2}\|\theta - \boldsymbol{\rho}^t + \boldsymbol{u}_{\boldsymbol{N}}^t\|^2$$

$$\boldsymbol{\rho}^{t+1} = \underset{\theta}{\operatorname{argmin}}\, \boldsymbol{\psi}(\theta) + (N\tau/2)\|\theta - \bar{\boldsymbol{\theta}}^{t+1} - \bar{\boldsymbol{u}}^t\|^2$$

$$\boldsymbol{u}_{\boldsymbol{i}}^{t+1} = \boldsymbol{u}_{\boldsymbol{i}}^t + \boldsymbol{\theta}_{\boldsymbol{i}}^{t+1} - \boldsymbol{\rho}^{t+1}, \boldsymbol{i} \leq \boldsymbol{N}$$

COCOA: A Dual Approach

$$\{1,\dots,n\} = \bigcup_{k=1}^{K} G_k, \;\; G_k \cap G_{k'} = \emptyset.$$

$$\begin{aligned} D(y) &= \frac{1}{n}\sum_{i=1}^{n} f_i^*(y_i) + \psi^*\left(-\frac{1}{n}Ay\right) \\ &= \frac{1}{n}\sum_{k=1}^{K} \underbrace{\left(\sum_{i\in G_k} f_i^*(y_i)\right)}_{\text{Divided into } K \text{ groups}} + \underbrace{\psi^*\left(-\frac{1}{n}\sum_{k=1}^{K} A_{G_k} y_{G_k}\right)}_{\text{needs synchronization}} \end{aligned}$$

source: T. Suzuki

COCOA: A Dual Approach

Samples divided into subsets $\{1,\ldots,n\} = \bigcup_{k=1}^{K} G_k, \;\; G_k \cap G_{k'} = \emptyset.$

$$D(y) = \frac{1}{n}\sum_{i=1}^{n} f_i^*(y_i) + \psi^*\left(-\frac{1}{n}Ay\right)$$

$$= \frac{1}{n}\sum_{k=1}^{K} \underbrace{\left(\sum_{i\in G_k} f_i^*(y_i)\right)}_{\text{Divided into } K \text{ groups}} + \underbrace{\psi^*\left(-\frac{1}{n}\sum_{k=1}^{K} A_{G_k} y_{G_k}\right)}_{\text{needs synchronization}}$$

source: T. Suzuki

COCOA: Ex. Quadratic Reg.

$$\min_{w \in \mathbb{R}^d} \quad \left[P(\boldsymbol{w}) := \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \ell_i(\boldsymbol{w}^T \boldsymbol{x}_i) \right]$$

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad \left[D(\boldsymbol{\alpha}) := -\frac{\lambda}{2}\|A\boldsymbol{\alpha}\|^2 - \frac{1}{n}\sum_{i=1}^{n} \ell_i^*(-\alpha_i) \right]$$

Algorithm 1: COCOA: Communication-Efficient Distributed Dual Coordinate Ascent

Input: $T \geq 1$, scaling parameter $1 \leq \beta_K \leq K$ (default: $\beta_K := 1$).
Data: $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ distributed over K machines
Initialize: $\boldsymbol{\alpha}_{[k]}^{(0)} \leftarrow \mathbf{0}$ for all machines k, and $\boldsymbol{w}^{(0)} \leftarrow \mathbf{0}$
for $t = 1, 2, \dots, T$
  for ***all machines*** $k = 1, 2, \dots, K$ ***in parallel***
    $(\Delta\boldsymbol{\alpha}_{[k]}, \Delta\boldsymbol{w}_k) \leftarrow \text{LocalDualMethod}(\boldsymbol{\alpha}_{[k]}^{(t-1)}, \boldsymbol{w}^{(t-1)})$
    $\boldsymbol{\alpha}_{[k]}^{(t)} \leftarrow \boldsymbol{\alpha}_{[k]}^{(t-1)} + \frac{\beta_K}{K}\Delta\boldsymbol{\alpha}_{[k]}$
  end
  reduce $\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} + \frac{\beta_K}{K}\sum_{k=1}^{K}\Delta\boldsymbol{w}_k$
end

COCOA: Ex. Quadratic Reg.

Procedure A: LOCALDUALMETHOD: Dual algorithm for prob. (2) on a single coordinate block k

---

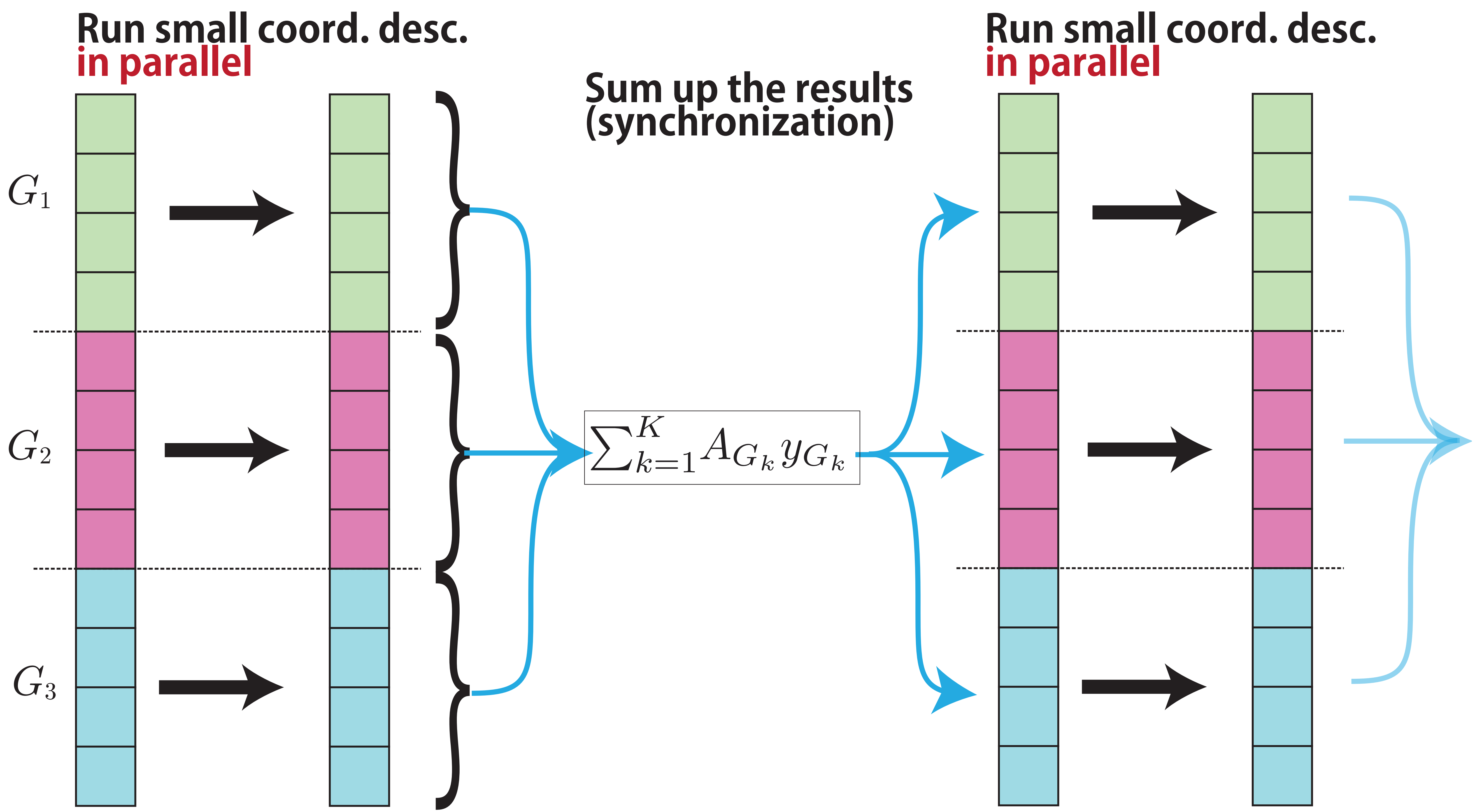
Input: Local $\boldsymbol{\alpha}_{[k]} \in \mathbb{R}^{n_k}$, and $\boldsymbol{w} \in \mathbb{R}^d$ consistent with other coordinate blocks of $\boldsymbol{\alpha}$ s.t. $\boldsymbol{w} = A\boldsymbol{\alpha}$
Data: Local $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n_k}$
Output: $\Delta\boldsymbol{\alpha}_{[k]}$ and $\Delta\boldsymbol{w} := A_{[k]}\Delta\boldsymbol{\alpha}_{[k]}$

Procedure B: LOCALSDCA: SDCA iterations for problem (2) on a single coordinate block k

---

Input: $H \geq 1$, $\boldsymbol{\alpha}_{[k]} \in \mathbb{R}^{n_k}$, and $\boldsymbol{w} \in \mathbb{R}^d$ consistent with other coordinate blocks of $\boldsymbol{\alpha}$ s.t. $\boldsymbol{w} = A\boldsymbol{\alpha}$
Data: Local $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n_k}$
Initialize: $\boldsymbol{w}^{(0)} \leftarrow \boldsymbol{w}$, $\Delta\boldsymbol{\alpha}_{[k]} \leftarrow \mathbf{0} \in \mathbb{R}^{n_k}$
for $h = 1, 2, \dots, H$
  choose $i \in \{1, 2, \dots, n_k\}$ *uniformly at random*
  find $\Delta\alpha$ *maximizing* $-\frac{\lambda n}{2}\|\boldsymbol{w}^{(h-1)} + \frac{1}{\lambda n}\Delta\alpha\, \boldsymbol{x}_i\|^2 - \ell_i^*\big(-(\alpha_i^{(h-1)} + \Delta\alpha)\big)$
  $\alpha_i^{(h)} \leftarrow \alpha_i^{(h-1)} + \Delta\alpha$
  $(\Delta\alpha_{[k]})_i \leftarrow (\Delta\alpha_{[k]})_i + \Delta\alpha$
  $\boldsymbol{w}^{(h)} \leftarrow \boldsymbol{w}^{(h-1)} + \frac{1}{\lambda n}\Delta\alpha\, \boldsymbol{x}_i$
end
Output: $\Delta\boldsymbol{\alpha}_{[k]}$ and $\Delta\boldsymbol{w} := A_{[k]}\Delta\boldsymbol{\alpha}_{[k]}$

COCOA


Run small coord. desc.
in parallel
Sum up the results
(synchronization)
Run small coord. desc.
in parallel
G_1
G_2
G_3
$\sum_{k=1}^{K} A_{G_k} y_{G_k}$

GPUs

Moore's Law

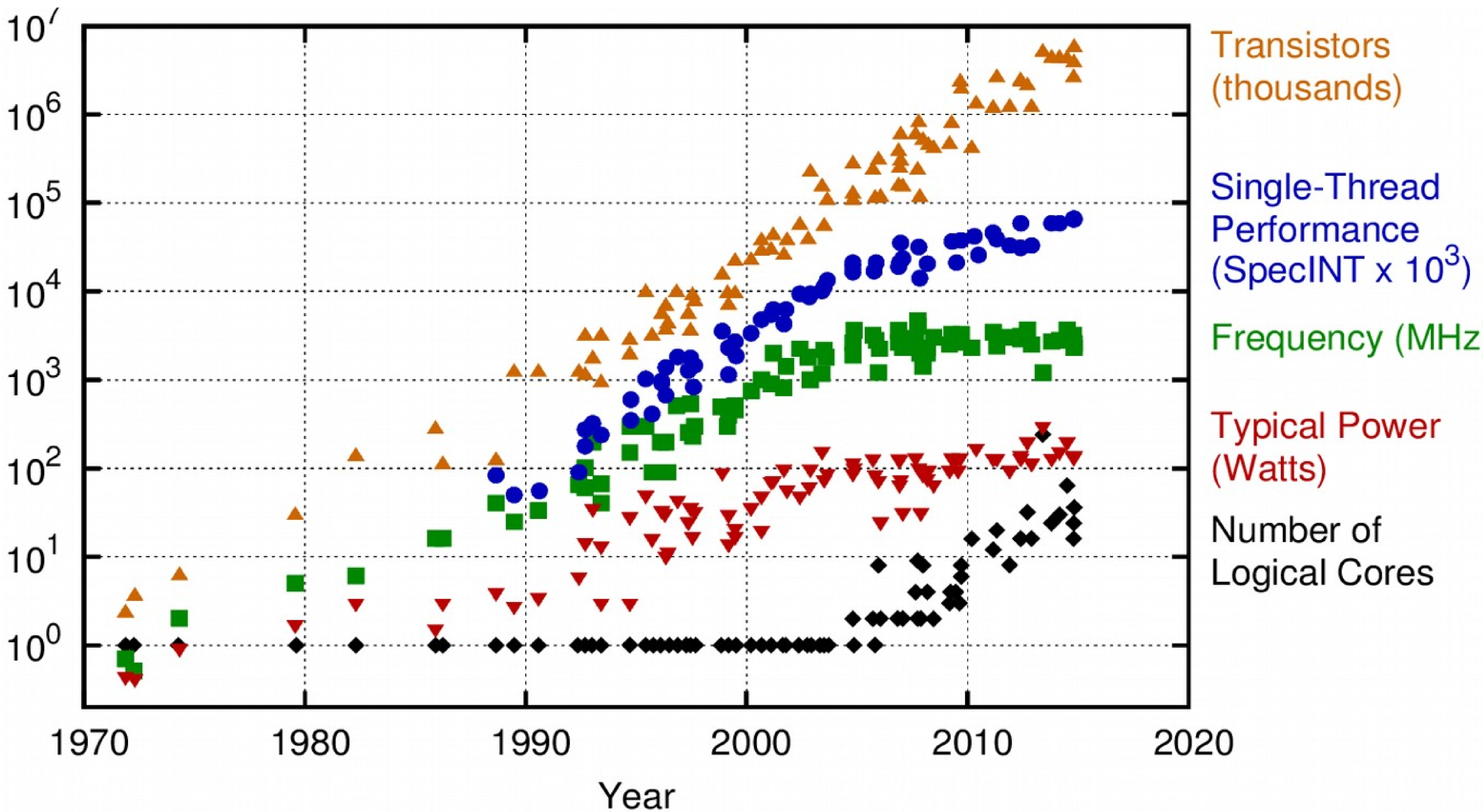
"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue"

Gordon Moore (Intel), 1965

"OK, maybe a factor of two every two years."

Gordon Moore (Intel), 1975 [paraphrased]

Moore's Law



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

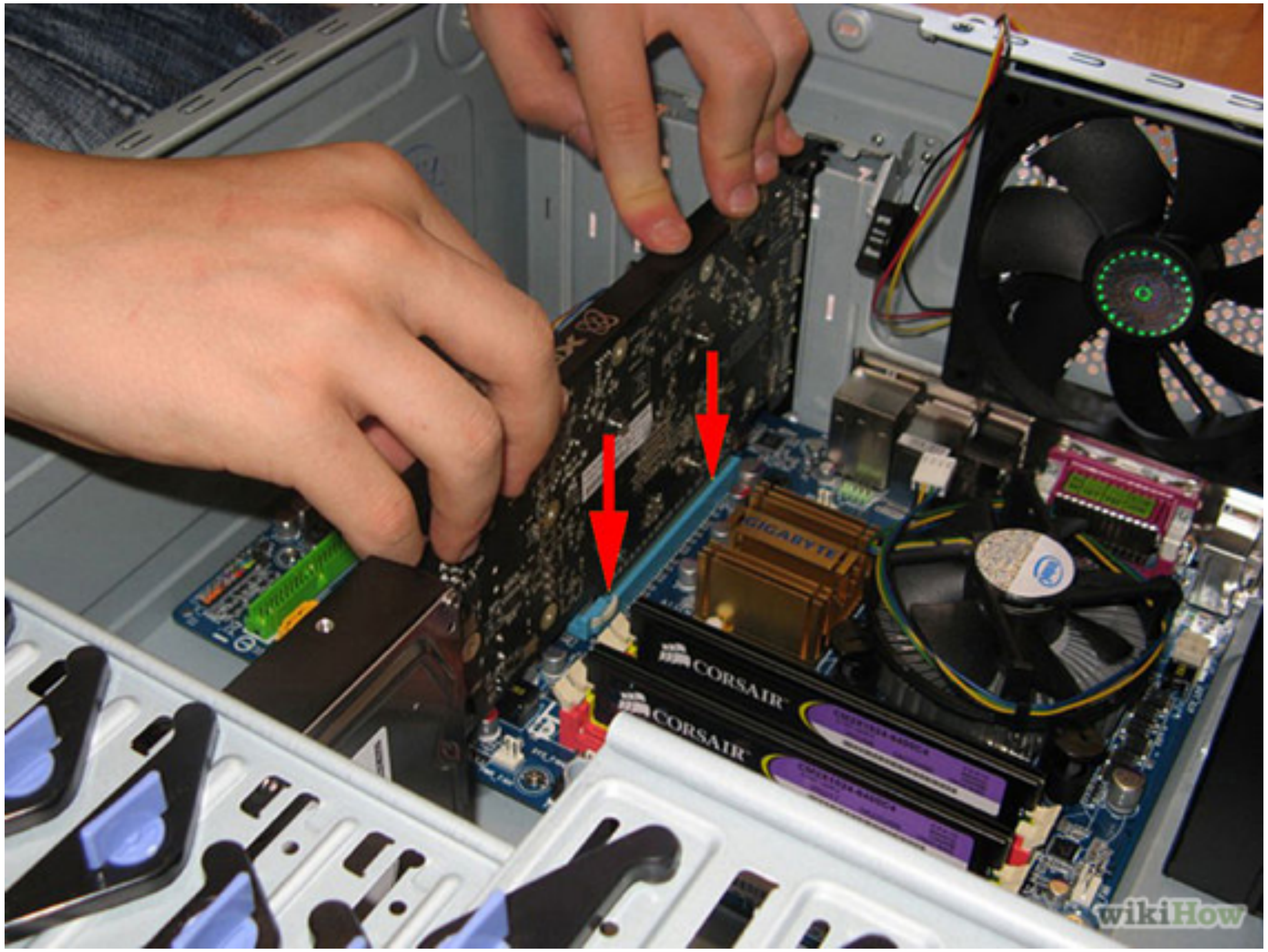
Solution: GPU

used to be a small piece of hardware…

GPU = Graphics Processing Unit

Solution: GPU

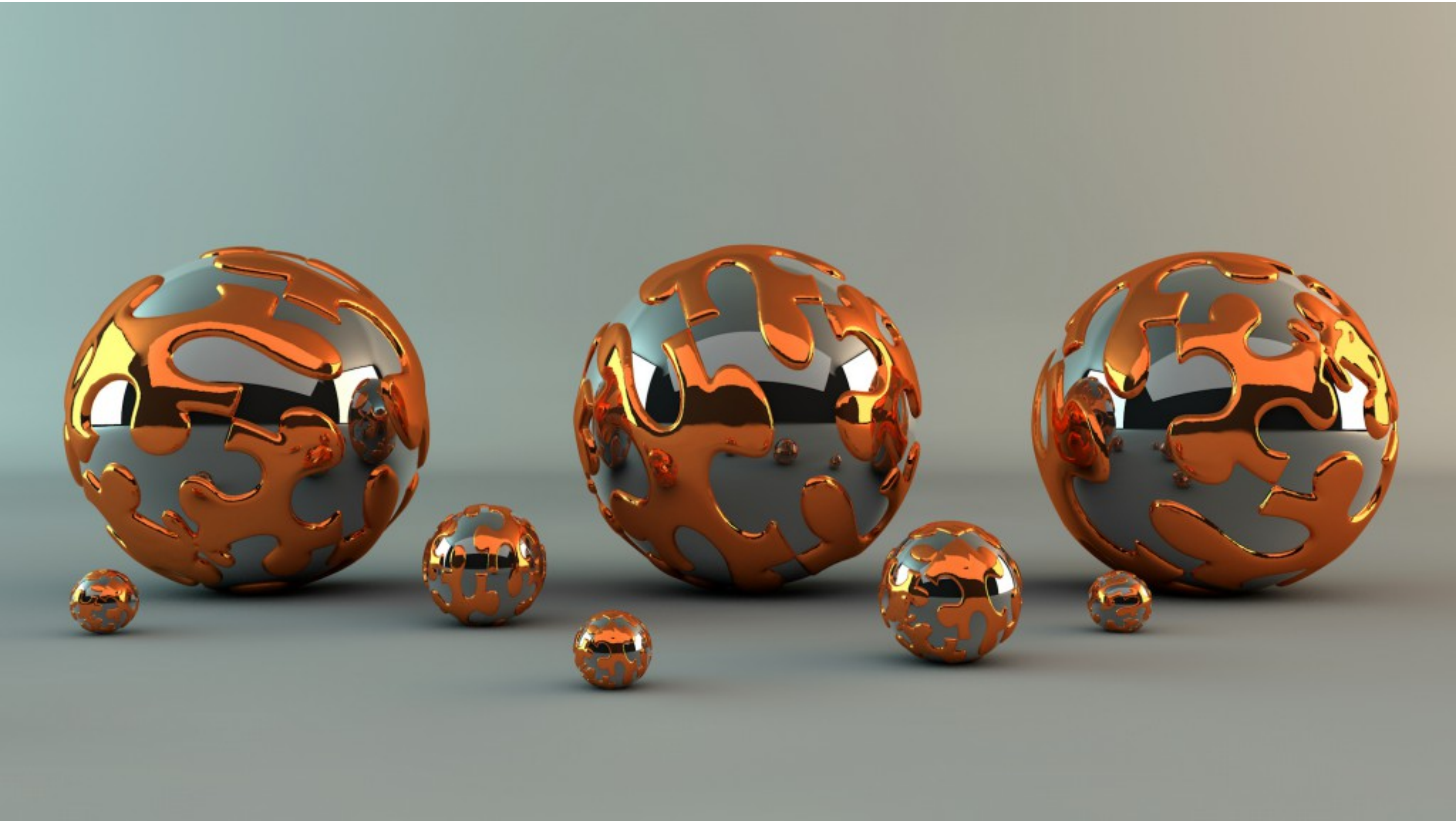
… plugged into computer, with video output…



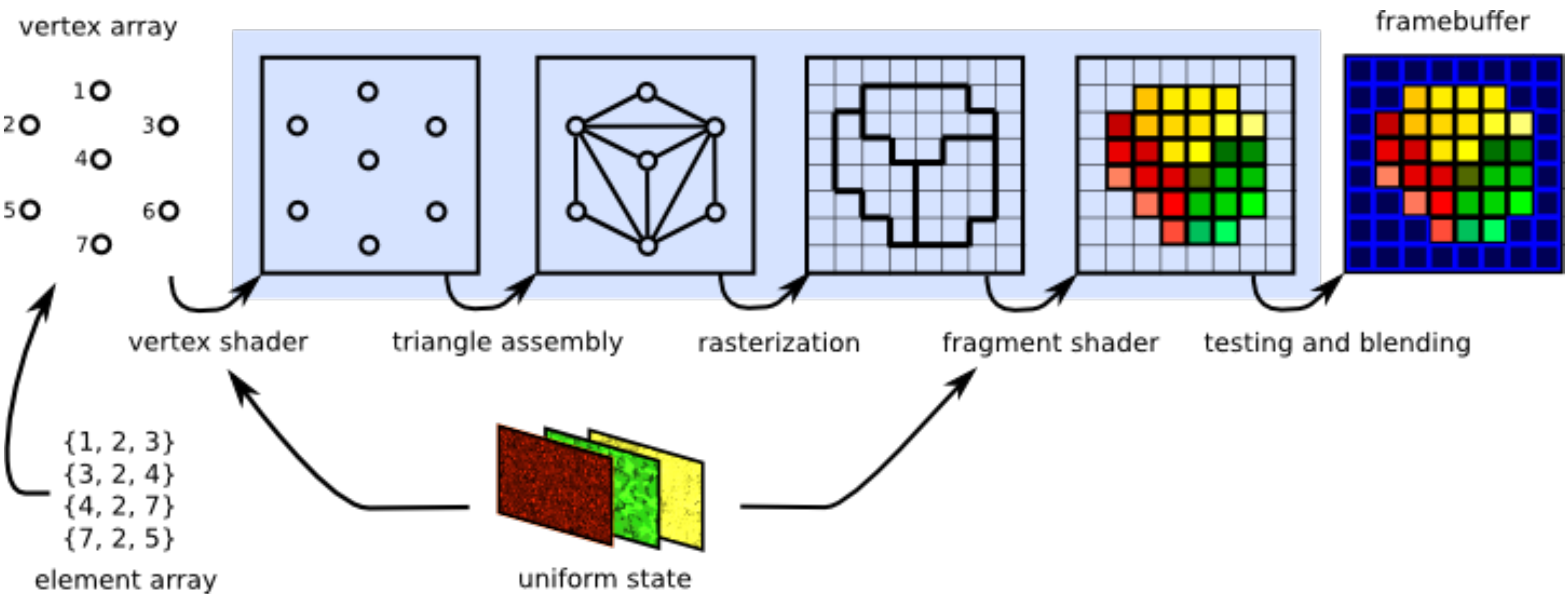
Solution: GPU

…of interest to gamers and video editors.

Graphics



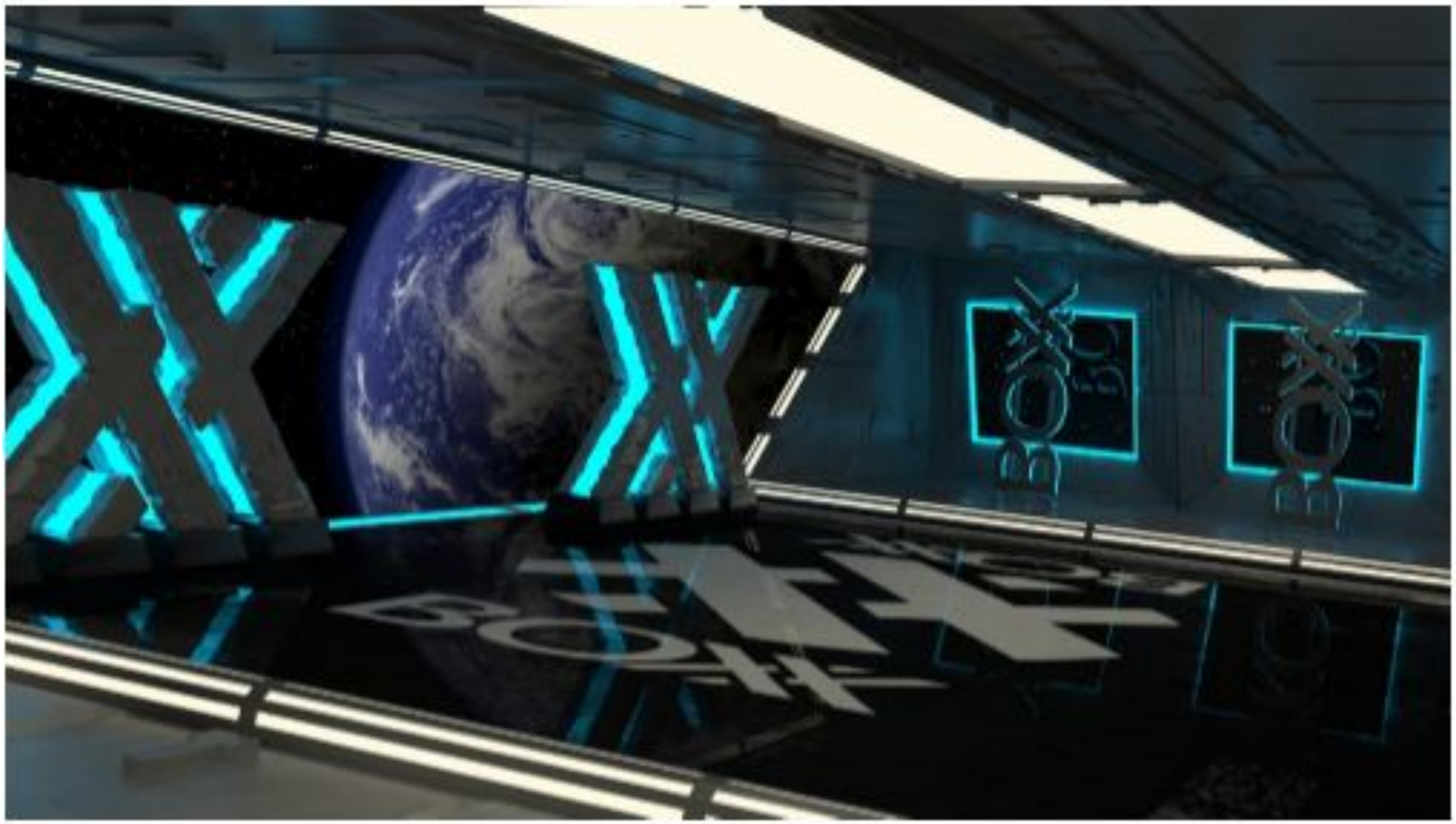
Graphics



vertex array
1
2
3
4
5
6
7
framebuffer
vertex shader
triangle assembly
rasterization
fragment shader
testing and blending
{1, 2, 3}
{3, 2, 4}
{4, 2, 7}
{7, 2, 5}
element array
uniform state

3D Rendering

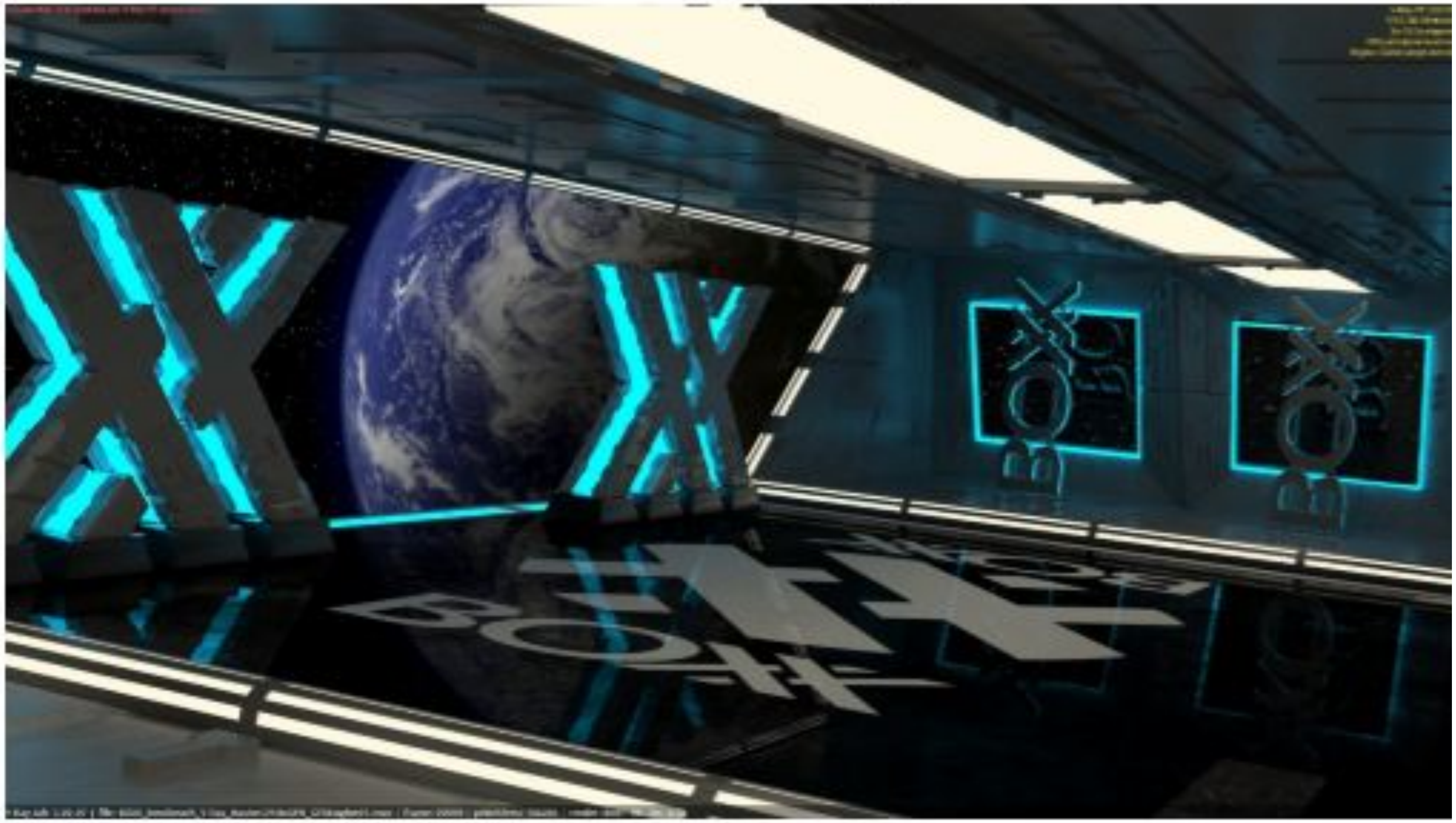
Rendered with V-Ray Advanced CPU



3.4 GHz 8 core Intel® Xeon®
Image Quality = 11.35
Render Time = 19 minutes 11 seconds

3D Rendering

Rendered with V-Ray RT GPU



High-end NVIDIA GPU with 2688 CUDA cores
Image Quality = 11.35
Render Time = 3 minutes 4 seconds

What are GPUs

Definition: GPU

A **programmable logic chip** (processor) specialized for **display functions.** The GPU renders images, animations and video for the computer's screen. GPUs are located on plug-in cards, in a chipset on the motherboard or in the same chip as the CPU.

A GPU performs parallel operations. Although it is used for 2D data as well as for zooming and panning the screen, a GPU is essential for smooth decoding and **rendering of 3D animations.**

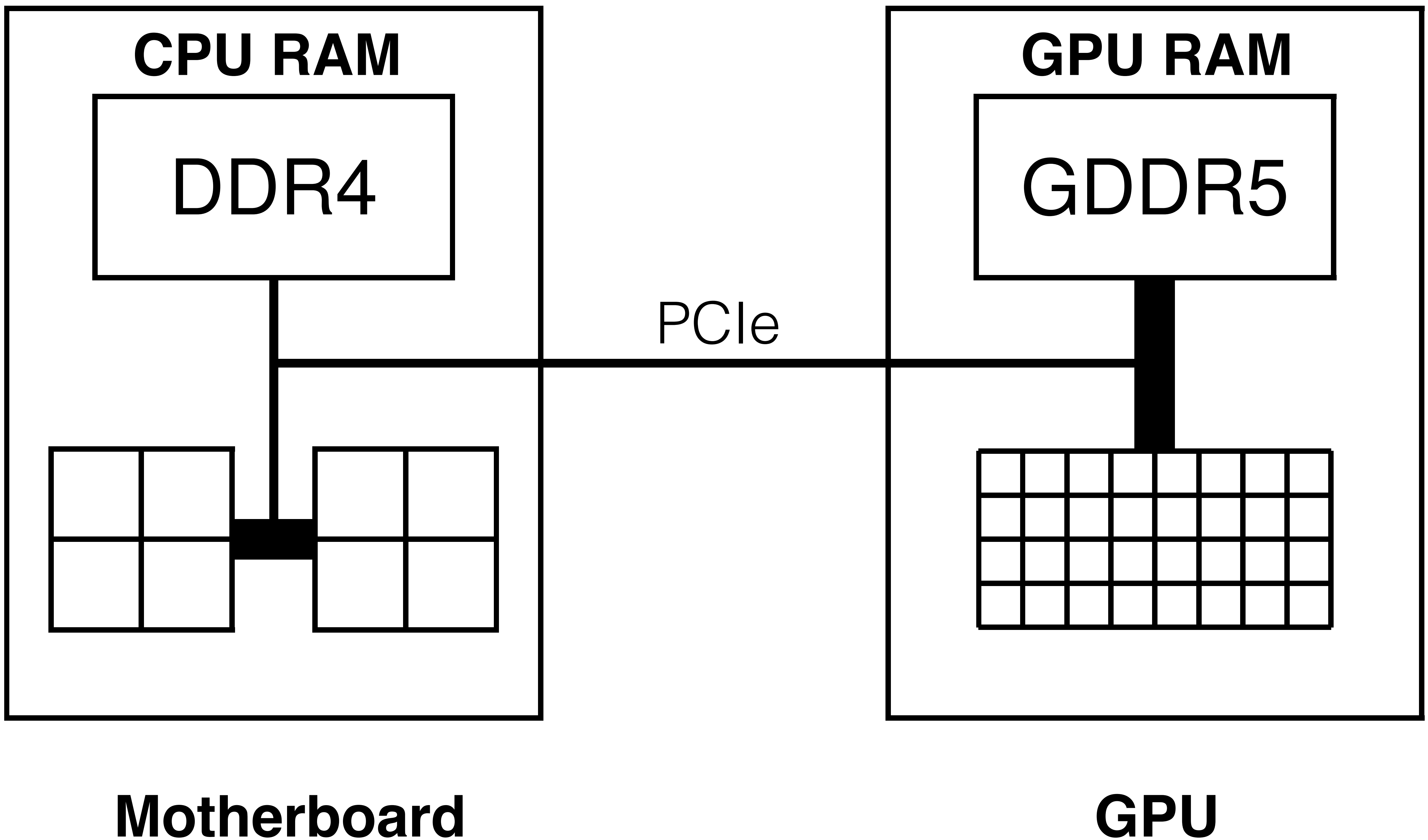
What are **GP**GPUs

Definition: **GP**GPU

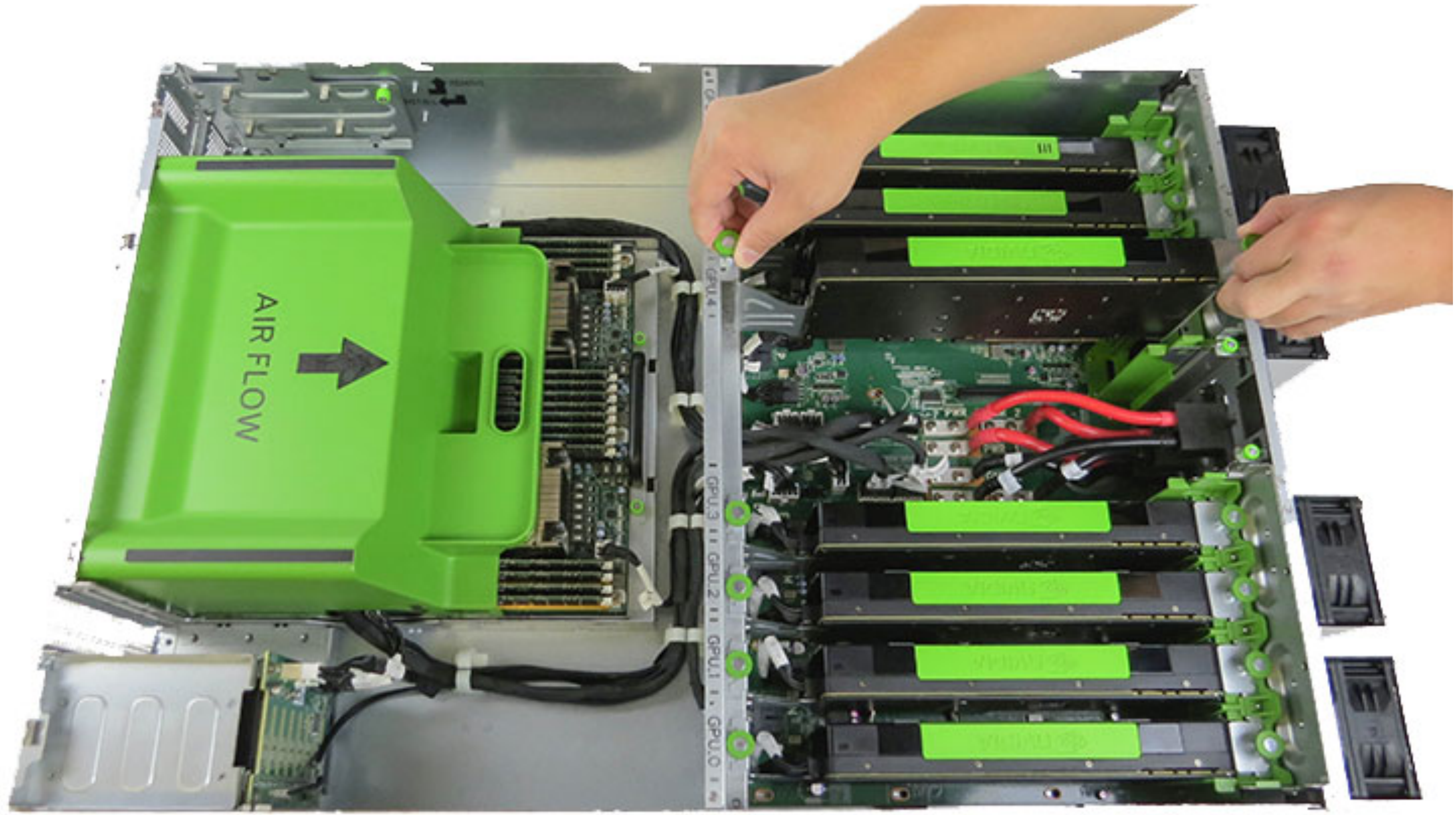
Using a GPU for general-purpose (**GP**) parallel processing applications rather than rendering images for the screen.

For fast results, applications such as sorting, **matrix algebra**, image processing and physical modeling require multiple sets of data to be processed in parallel.

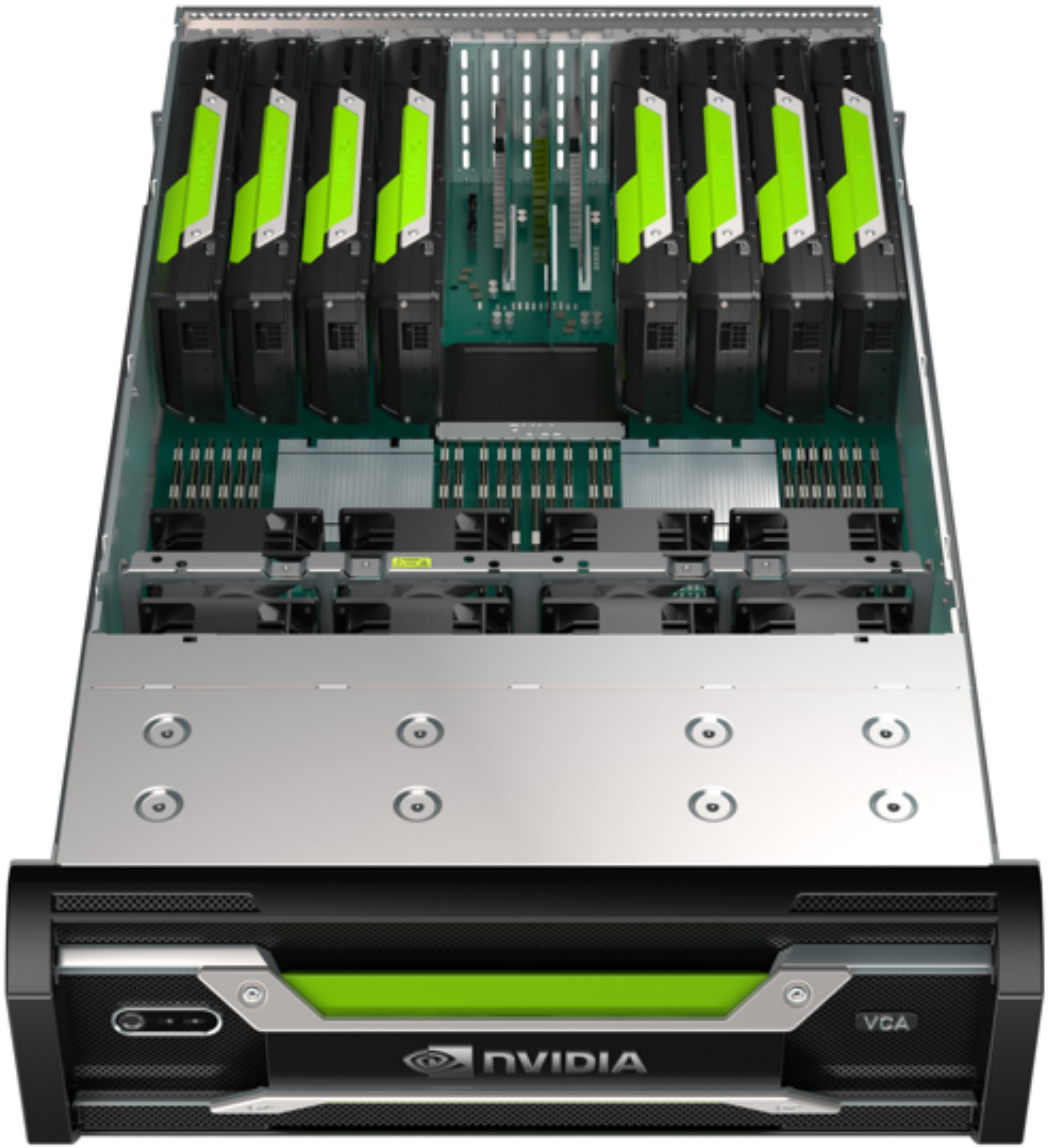
At very basic level...



In the real world



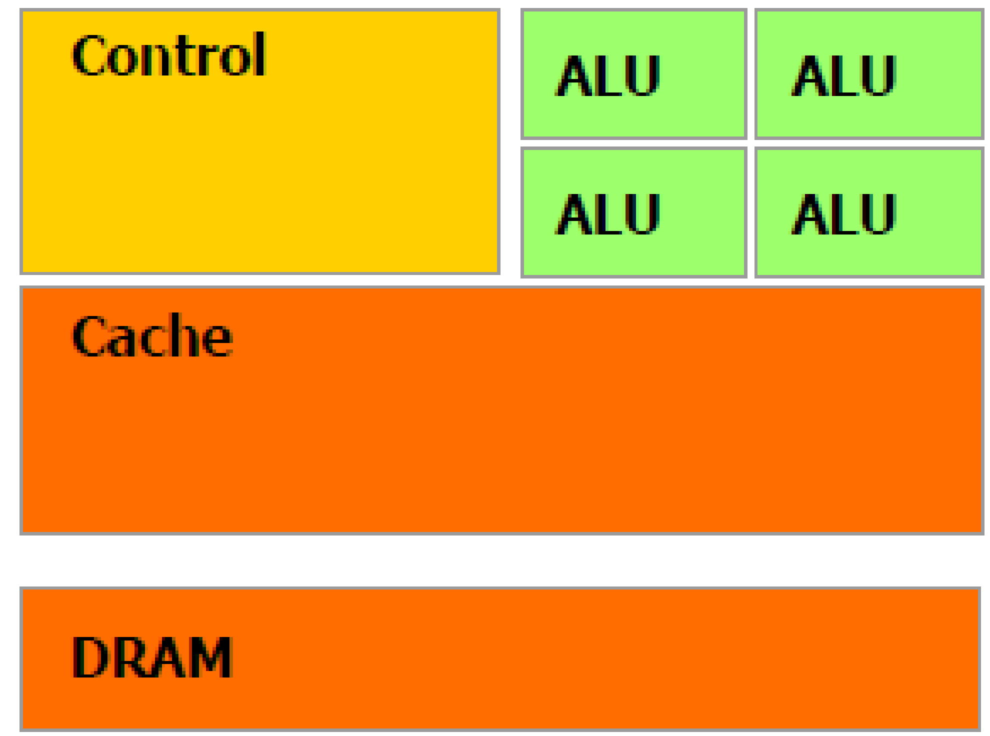
In the real world



CPU

Single Instructions, Multiple Data (SIMD)

large data-**caching**
large flow **control units**
few Arithmetic Logical Units (ALU, cores), but fast



Example: Intel Xeon E5-2670 CPU

8 cores (16 threads)
2.6 GHz
2.3 billion transistors
20 MB on chip cache
Flexible DRAM size

GPU

Single Instructions, Multiple Threads (SIMT)

small **cache, control flow**

Many ALUs (cores), slow.

Highly parallel.

Example: Kepler K20x GPU

2688 (14 x 192) cores

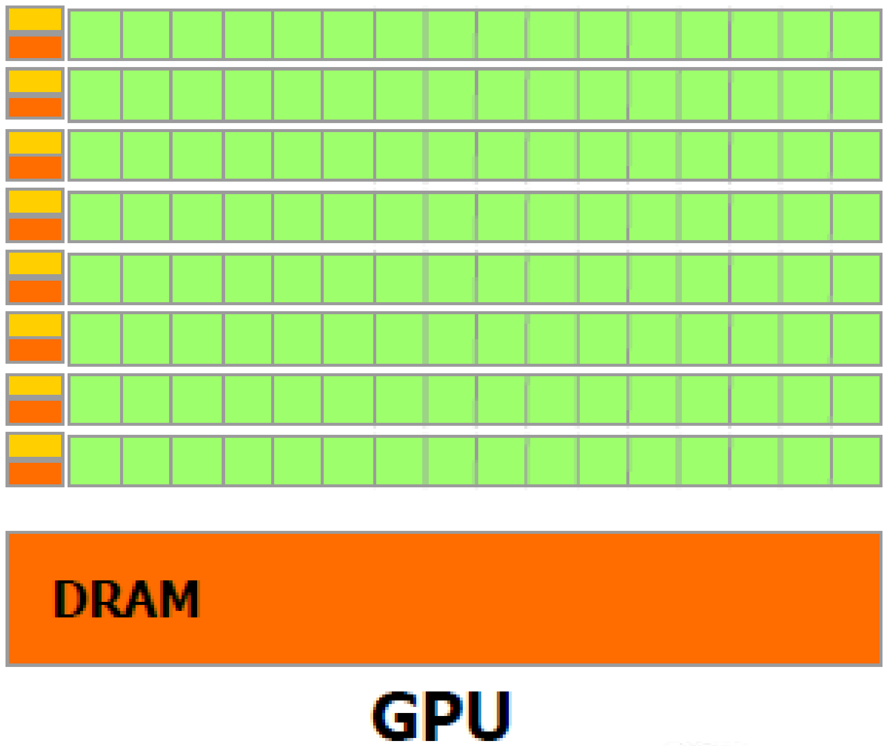
0.73 GHz

28nm features

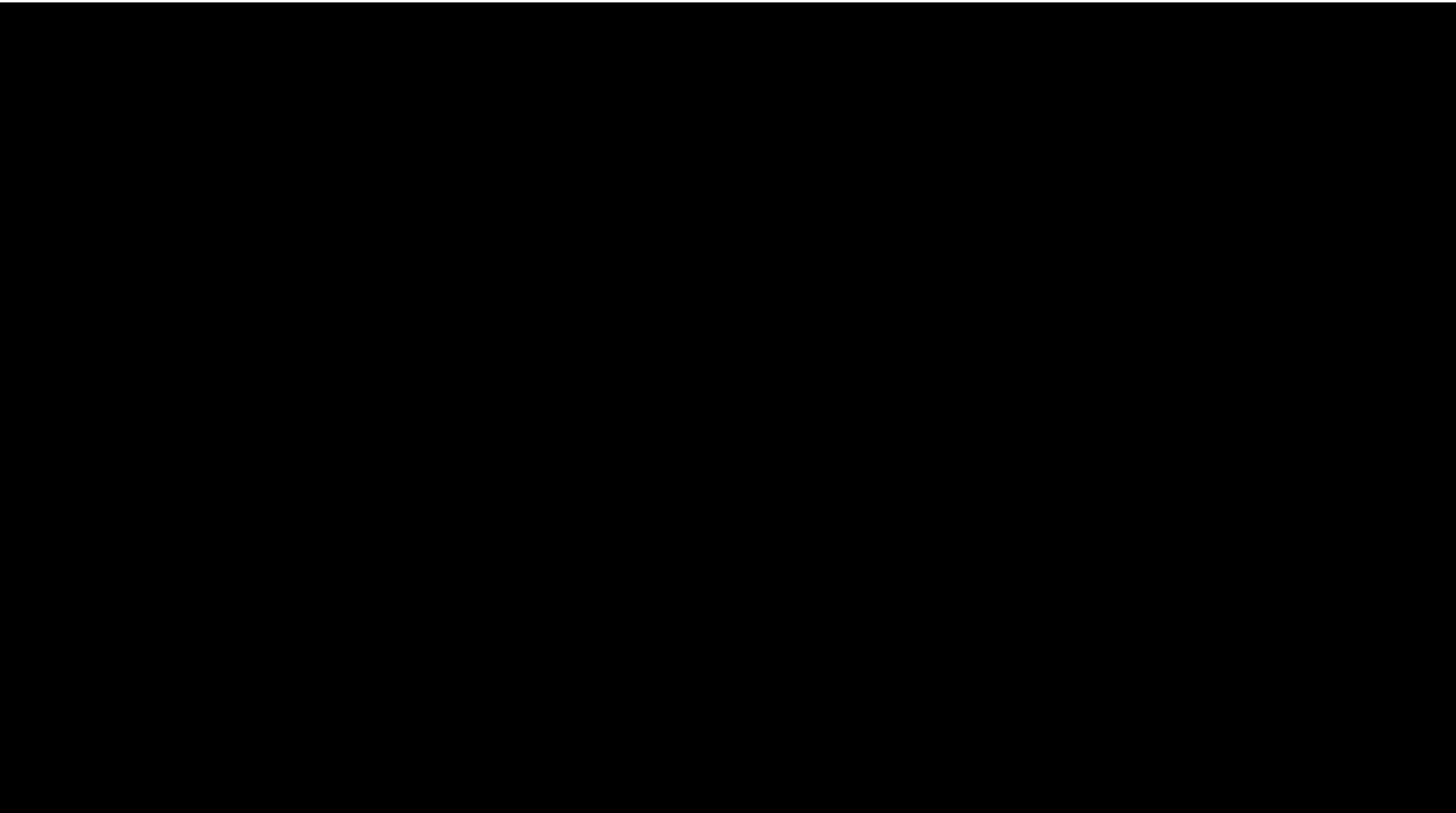
7.1 billion transistors

1.5 MB on-chip L2 cache

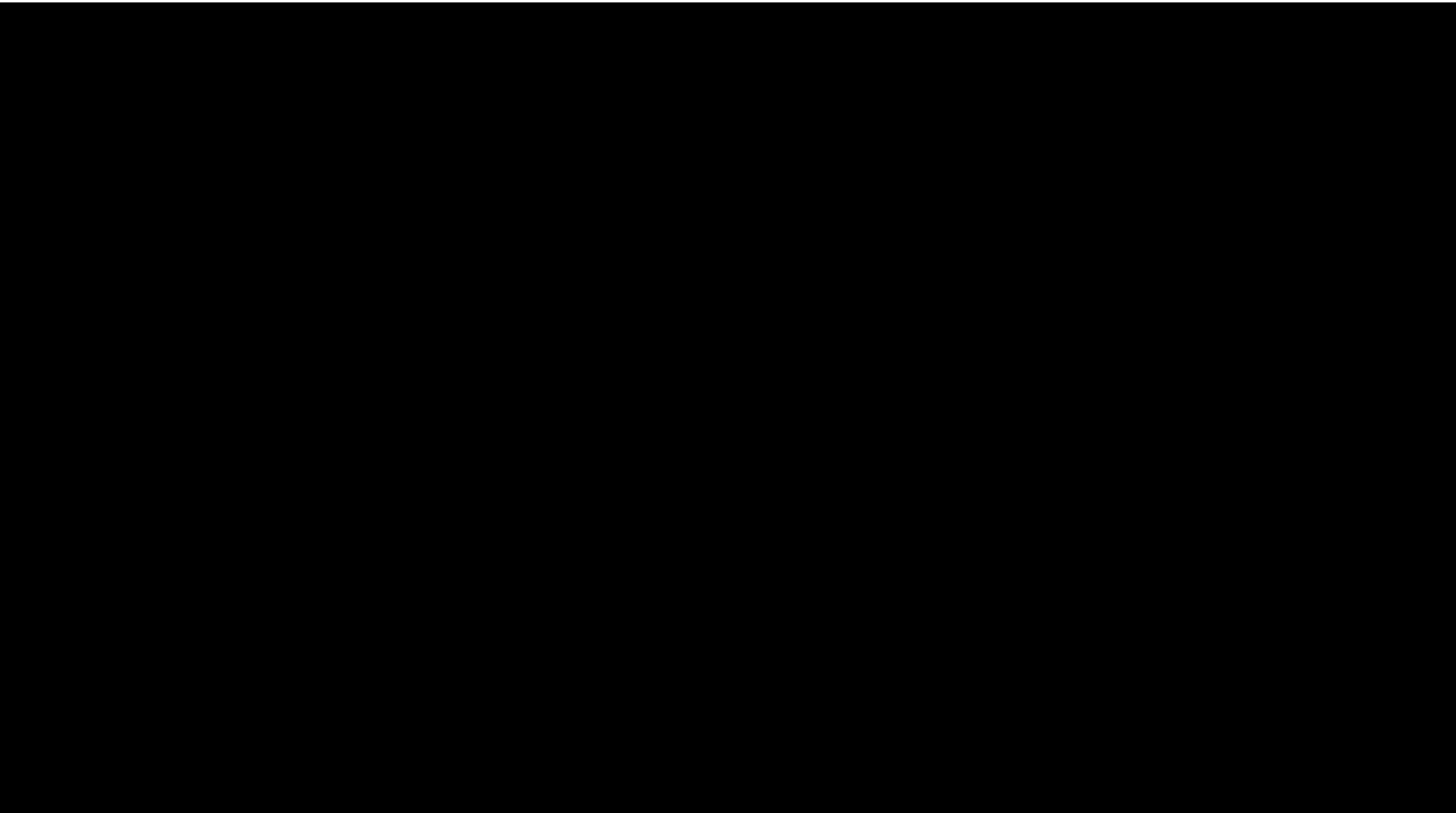
Only 6GB on chip memory



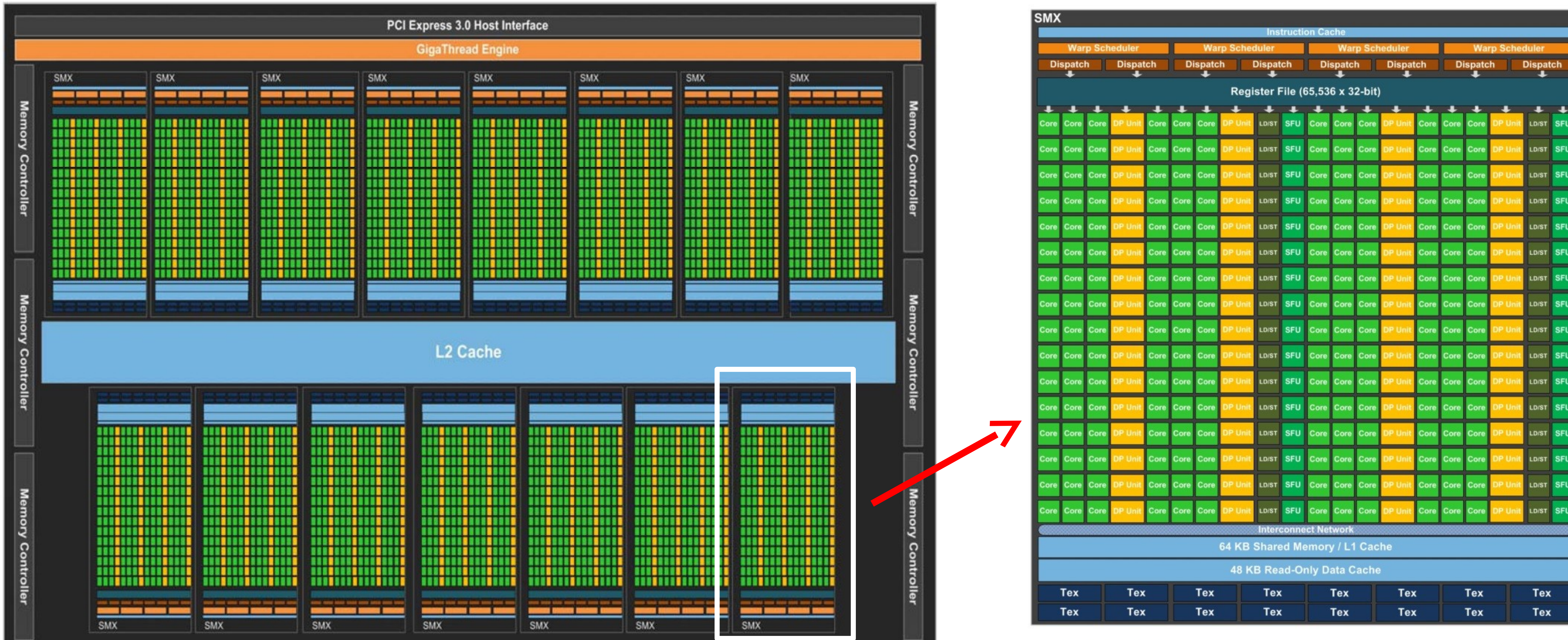
GPU *vs.* CPU



GPU *vs.* CPU



GPU Example: Kepler



SMX: 192 single-precision CUDA cores, 64 double-precision units, 32 special function units (SFU), and 32 load/store units (LD/ST).

Set of 14~15 SIMD Streaming Multiprocessors (SMX)

Each Multiprocessor has 192 cores, 64k L1 Cache.

Each SMX can handle up to 2000 threads.

GPU Example: Kepler

One SMX

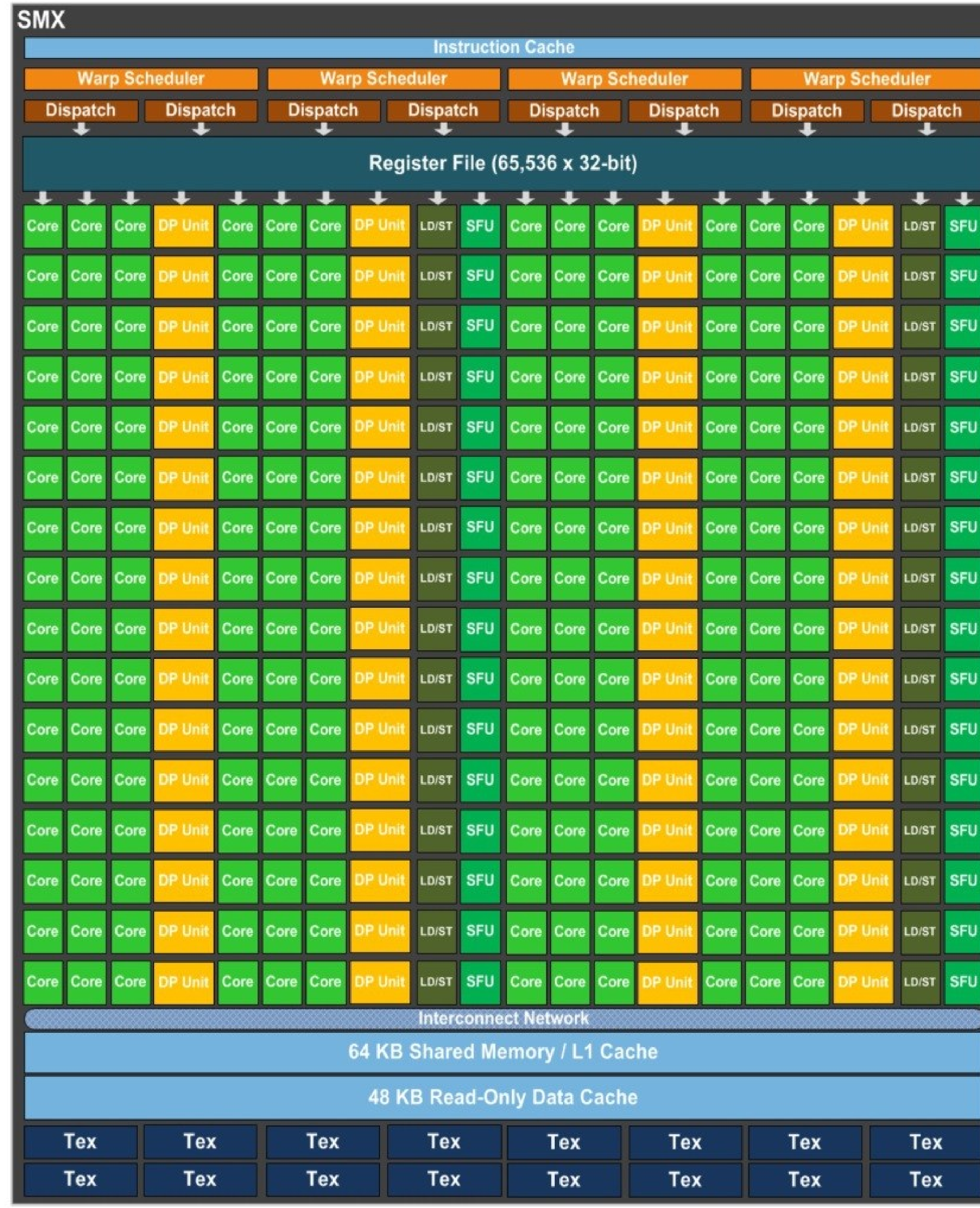
12 x 16=192 cores

32 Special Function Units

32 Load/Store Units

64 Double Precision Units

64k shared memory



SMX: 192 single-precision CUDA cores, 64 double-precision units, 32 special function units (SFU), and 32 load/store units (LD/ST).

GPU

GPU	G80	GT200	Fermi	Kepler
Transistors	681 million	1.4 billion	3.0 billion	7.0 billion
CUDA Cores	128	240	512 @ 1.15 GHz	2688 @ 0.73 GHz
Double Precision Floating Point Capability	None	30 FMA ops / clock	256 FMA ops /clock	1344 FMA ops/clock
Single Precision Floating Point Capability	128 MAD ops/clock	240 MAD ops / clock	512 FMA ops /clock	2688 FMA ops/clock
Special Function Units (SFUs) / SM	2	2	4	32
Warp schedulers (per SM)	1	1	2	2
Shared Memory (per SM)	16 KB	16 KB	Configurable 48 KB or 16 KB	Configurable 48 KB, 16 KB or 32 KB
L1 Cache (per SM)	None	None	Configurable 16 KB or 48 KB	Configurable 48 KB, 16 KB or 32 KB
L2 Cache	None	None	768 KB	1.5 MB
ECC Memory Support	No	No	Yes	Yes
Concurrent Kernels	No	No	Up to 16	Up to 32 + Dyn. Parallel
Load/Store Address Width	32-bit	32-bit	64-bit	64-bit

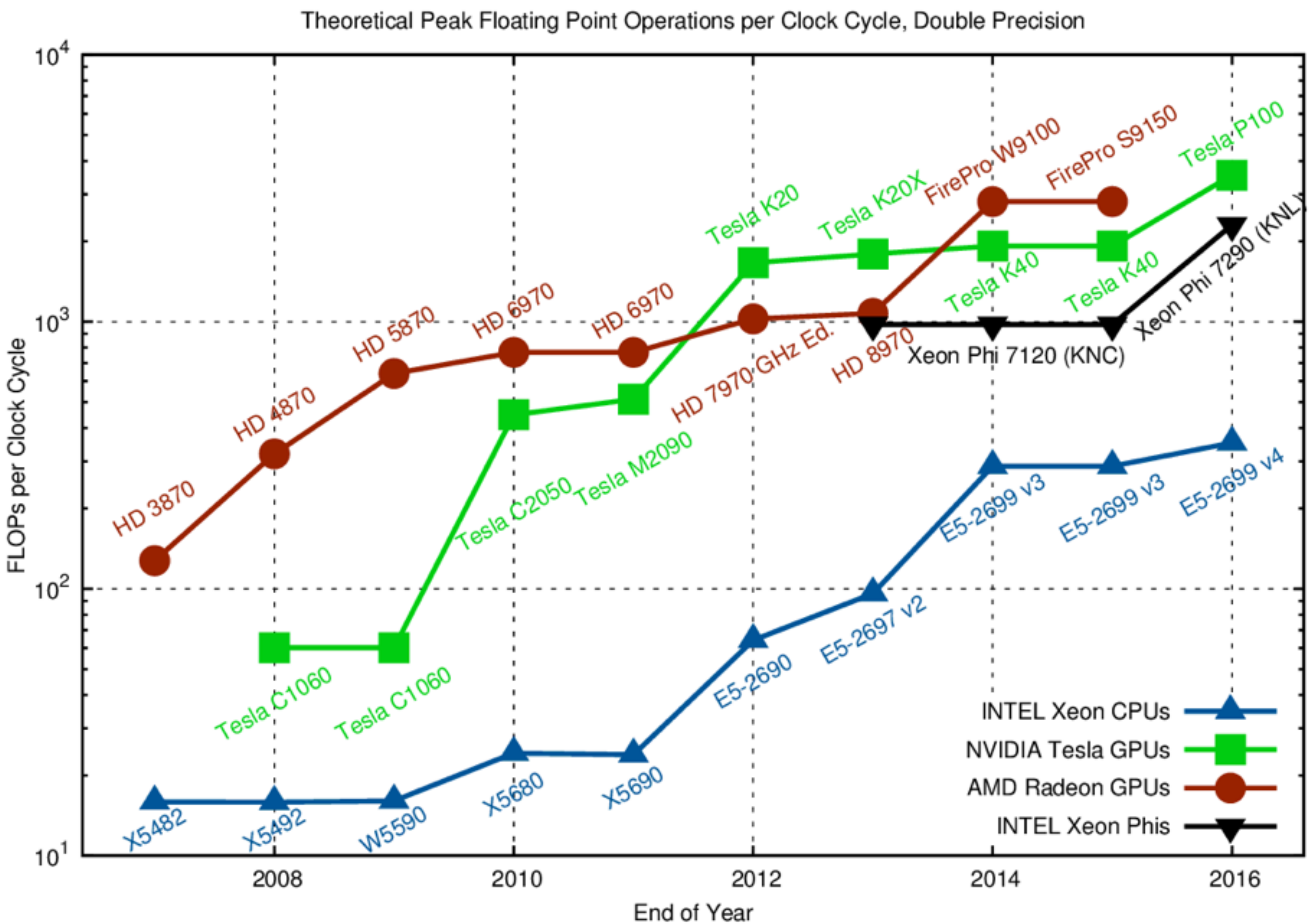
GPU

Because GPUs were designed to apply the same shading function to many pixels simultaneously, GPUs can be used to apply the same **simple** *function to many data points simultaneously*

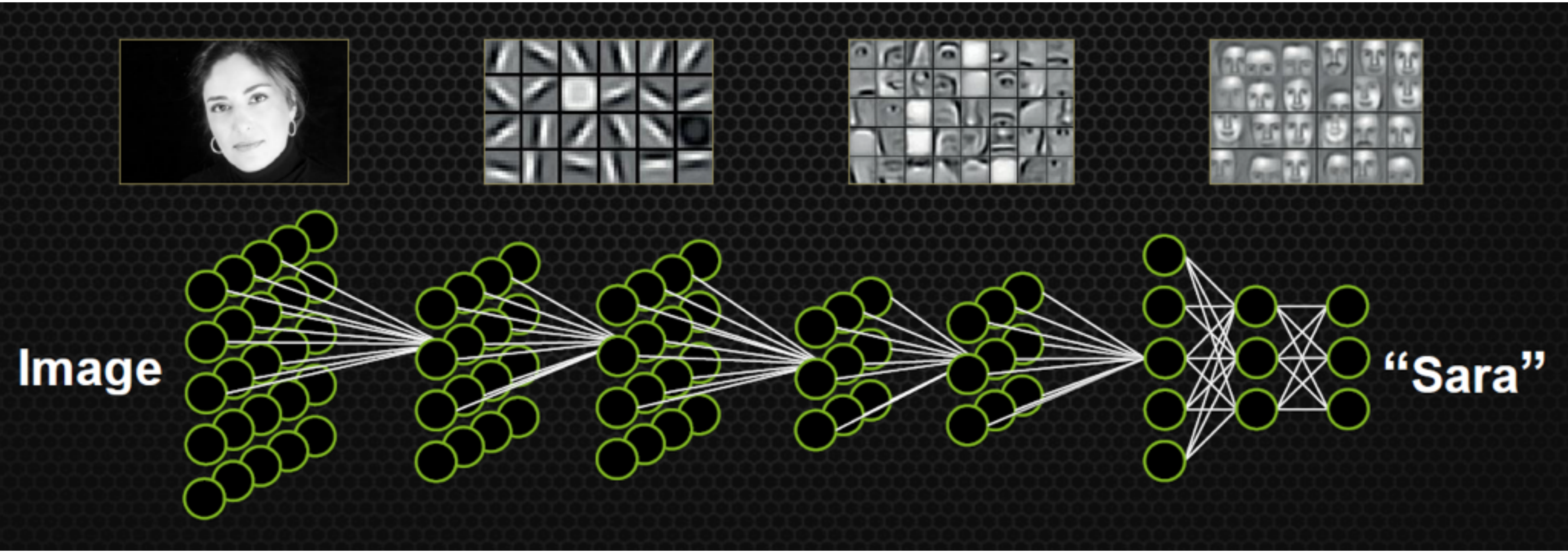
How simple?

Essentially, matrix algebra and special functions on each element (exp, log, sin etc...)

How fast?


Theoretical Peak Floating Point Operations per Clock Cycle, Double Precision
FLOPs per Clock Cycle
End of Year
2008
2010
2012
2014
2016
10^1
10^2
10^3
10^4
HD 3870
HD 4870
HD 5870
HD 6970
HD 6970
HD 7970 GHz Ed.
HD 8970
FirePro W9100
FirePro S9150
Tesla C1060
Tesla C1060
Tesla C2050
Tesla M2090
Tesla K20
Tesla K20X
Tesla K40
Tesla K40
Tesla P100
Xeon Phi 7120 (KNC)
Xeon Phi 7290 (KNL)
X5482
X5492
W5590
X5680
X5690
E5-2690
E5-2697 v2
E5-2699 v3
E5-2699 v3
E5-2699 v4
INTEL Xeon CPUs
NVIDIA Tesla GPUs
AMD Radeon GPUs
INTEL Xeon Phis

Crucial for Deep Learning



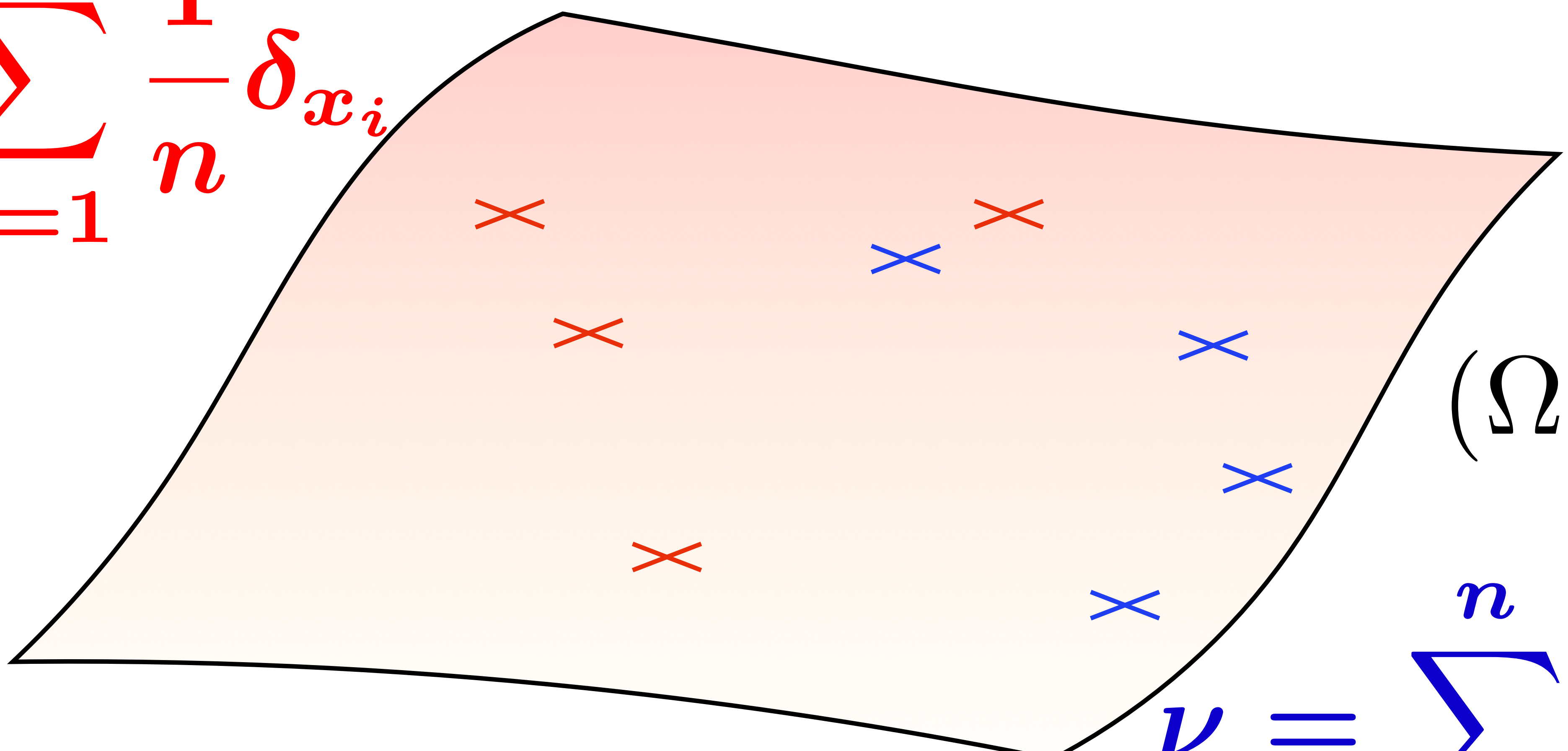
Why?

Multilayer Neural Networks only use element wise operations (hinge, softmax, tanh, sigmoid) and matrix products, exactly those operations that GPU are good for.

A more concrete Math Problem.

Optimal Assignment Problem

$$\mu = \sum_{i=1}^{n} \frac{1}{n} \delta_{x_i}$$



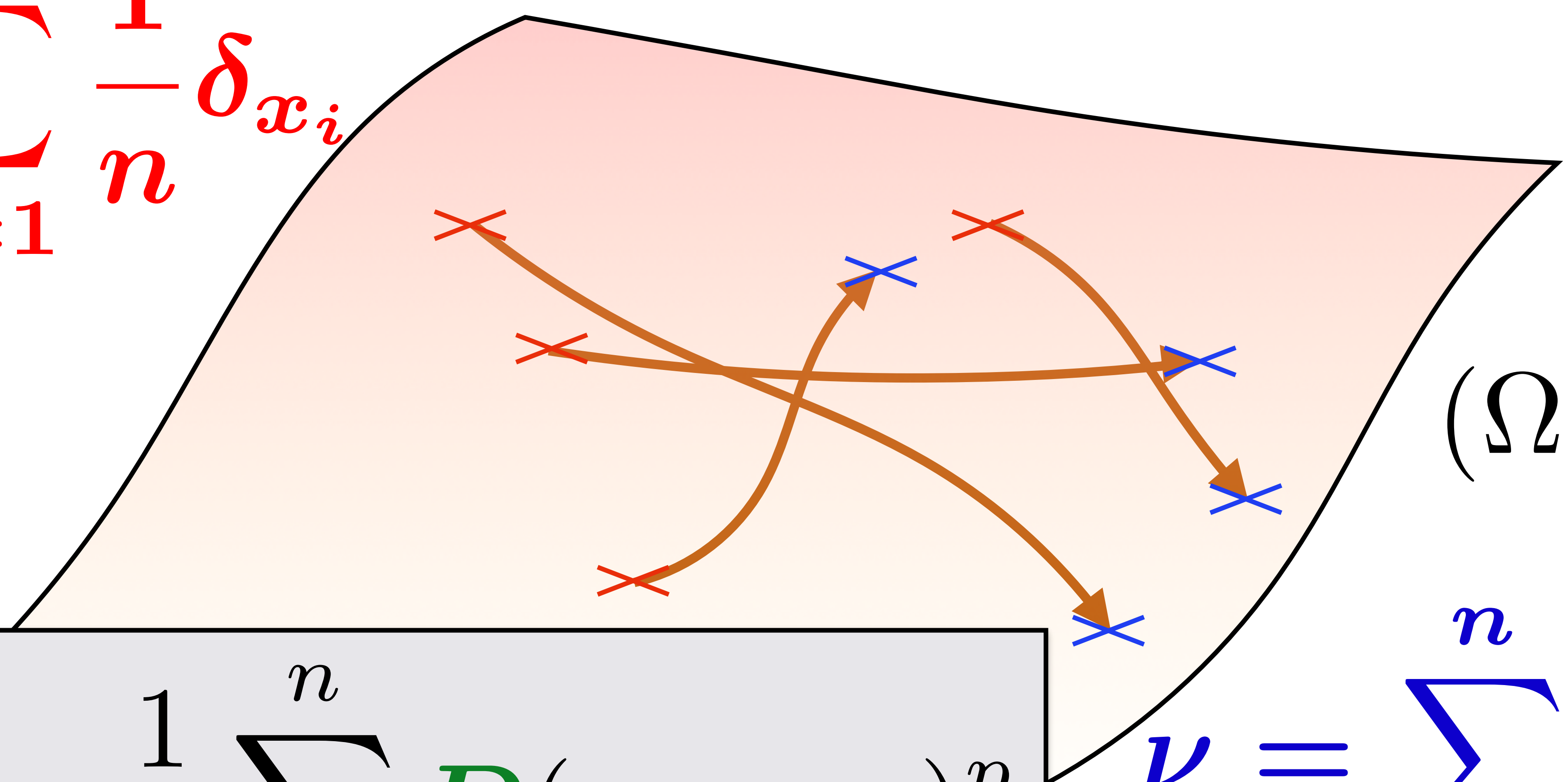
$$(\Omega, D)$$

$$\nu = \sum_{j=1}^{n} \frac{1}{n} \delta_{y_j}$$

A more concrete Math Problem.

Optimal Assignment Problem

$$\mu = \sum_{i=1}^{n} \frac{1}{n} \delta_{x_i}$$



$$(\Omega, D)$$

$$\nu = \sum_{j=1}^{n} \frac{1}{n} \delta_{y_j}$$

$$C(\sigma) = \frac{1}{n} \sum_{i=1}^{n} D(x_i, y_{\sigma_i})^p$$

Optimal Assignment Problem

$$\boxed{\mathrm{OA}(\boldsymbol{\mu}, \boldsymbol{\nu}) = \min_{\boldsymbol{\sigma} \in S_n} C(\boldsymbol{\sigma})}$$

$$M_{\mathbf{X}\mathbf{Y}} \stackrel{\text{def}}{=} [D(\boldsymbol{x}_i, \boldsymbol{y}_j)^p]_{ij}$$

$$P_{\boldsymbol{\sigma}} = [\mathbf{1}_{\boldsymbol{\sigma}_i = j}/n]_{i,j}$$

$$\boxed{\min_{\boldsymbol{\sigma} \in S_n} C(\boldsymbol{\sigma}) = \min_{\boldsymbol{\sigma} \in S_n} \langle P_{\boldsymbol{\sigma}}, M_{\mathbf{X}\mathbf{Y}} \rangle}$$

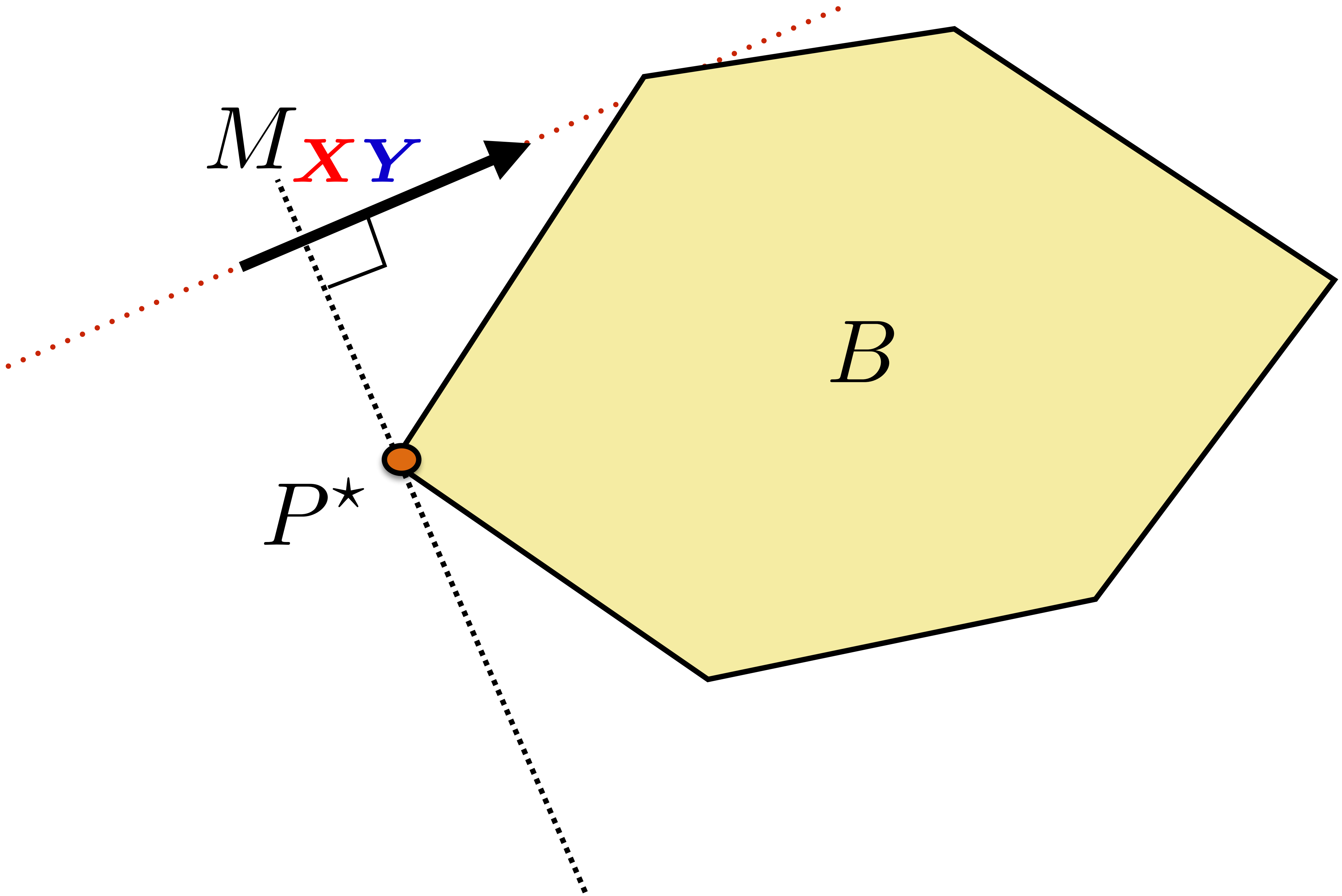
Optimal Assignment Problem

$$\min_{\boldsymbol{\sigma} \in S_n} C(\boldsymbol{\sigma}) = \min_{\boldsymbol{\sigma} \in S_n} \langle P_{\boldsymbol{\sigma}}, M_{\mathbf{X}\mathbf{Y}} \rangle$$

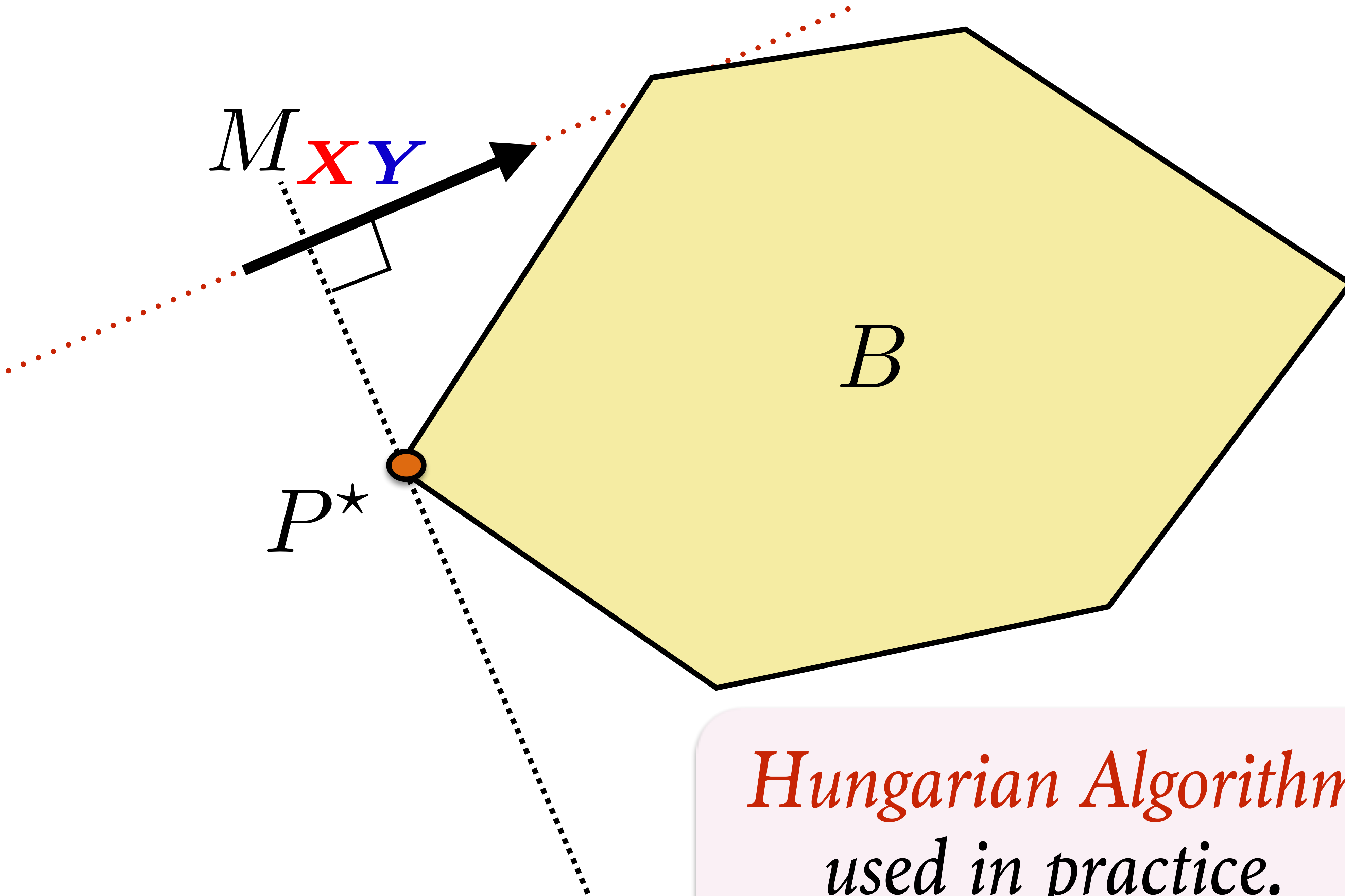
$$B = \left\{ P \in \mathbb{R}_+^{n \times n} | P\mathbf{1} = P^T\mathbf{1} = \frac{\mathbf{1}}{n} \right\}$$

$$\mathrm{OA}(\boldsymbol{\mu}, \boldsymbol{\nu}) = \min_{\boldsymbol{P} \in B} \langle \boldsymbol{P}, M_{\mathbf{X}\mathbf{Y}} \rangle$$

Optimal Assignment



Optimal Assignment



Hungarian Algorithm
used in practice.

Solving OA using Matrix Products

$$\mathrm{OA}_\gamma(\boldsymbol{\mu},\boldsymbol{\nu}) = \min_{\boldsymbol{P}\in B} \langle \boldsymbol{P}, M_{\mathbf{XY}}\rangle - \gamma E(\boldsymbol{P})$$

$$E(P) \stackrel{\text{def}}{=} -\sum_{i,j=1}^{n} P_{ij}(\log P_{ij})$$

Solving OA using Matrix Products

$$\mathrm{OA}_\gamma(\boldsymbol{\mu},\boldsymbol{\nu}) = \min_{\boldsymbol{P}\in B} \langle \boldsymbol{P}, M_{\boldsymbol{XY}}\rangle - \gamma E(\boldsymbol{P})$$

$$E(P) \stackrel{\text{def}}{=} -\sum_{i,j=1}^{n} P_{ij}(\log P_{ij})$$

$$L(P,\alpha,\beta) = \sum_{ij} P_{ij}M_{ij} + \gamma P_{ij}\log P_{ij} + \alpha^T(P\mathbf{1}-\mathbf{1}/n) + \beta^T(P^T\mathbf{1}-\mathbf{1}/n)$$

$$\partial L/\partial P_{ij} = M_{ij} + \gamma(\log P_{ij}+1) + \alpha_i + \beta_j$$

$$(\boldsymbol{\partial L/\partial P_{ij}} = 0) \Rightarrow P_{ij} = e^{\frac{\alpha_i}{\gamma}+\frac{1}{2}}\, e^{-\frac{M_{ij}}{\gamma}}\, e^{\frac{\beta_j}{\gamma}+\frac{1}{2}} = \boldsymbol{u_i}\, K_{ij}\boldsymbol{v_j}$$

Solving OA using Matrix Products

$$\mathrm{OA}(\boldsymbol{\mu}, \boldsymbol{\nu}) = \min_{\boldsymbol{P} \in B} \langle \boldsymbol{P}, M_{\mathbf{XY}} \rangle$$

Hungarian Algorithm

Cubic complexity

$$\mathrm{OA}_\gamma(\boldsymbol{\mu}, \boldsymbol{\nu}) = \min_{\boldsymbol{P} \in B} \langle \boldsymbol{P}, M_{\mathbf{XY}} \rangle - \gamma E(\boldsymbol{P})$$

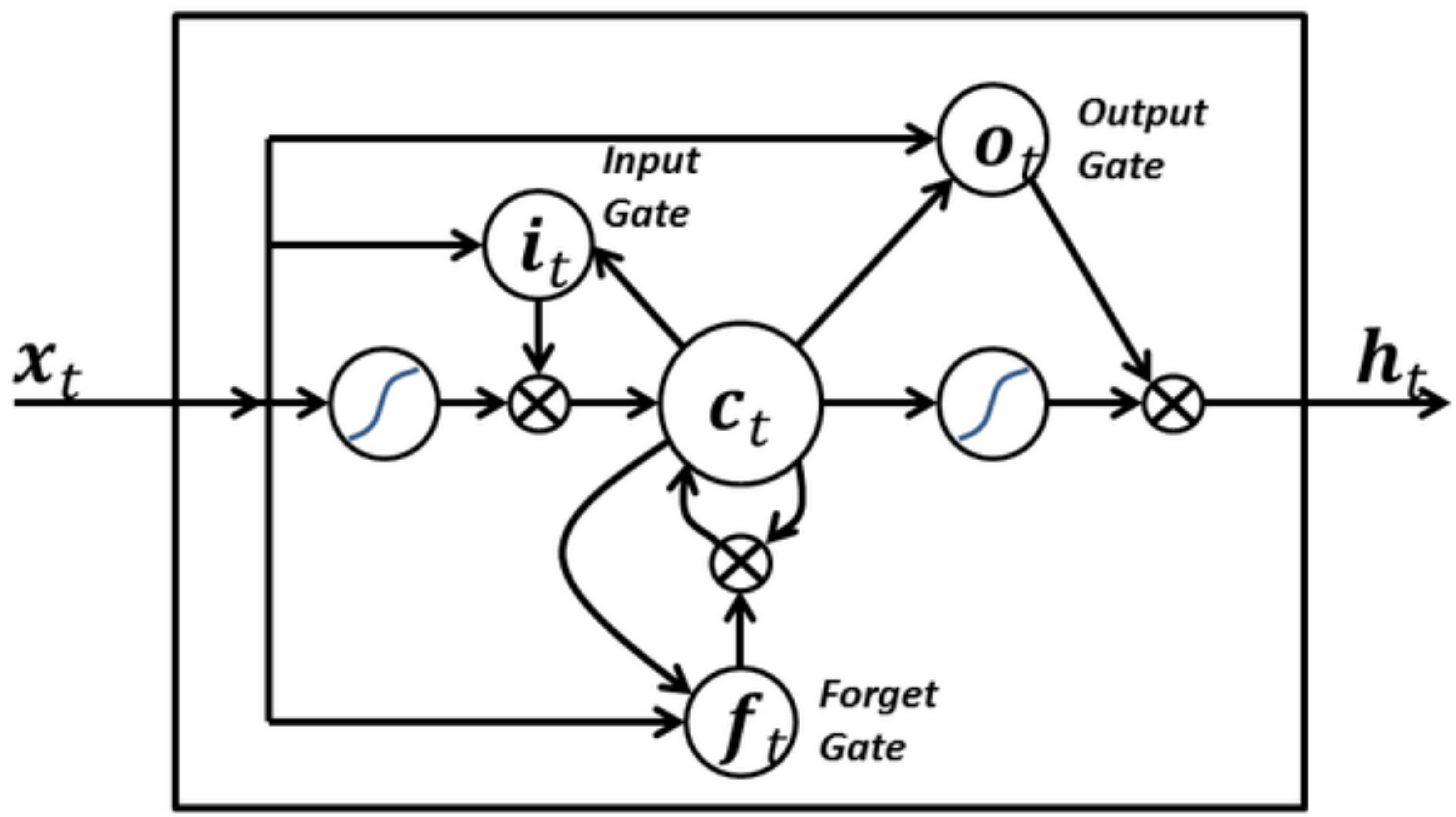
$$\boldsymbol{P}^* = D(\boldsymbol{u}) K D(\boldsymbol{v}); \boldsymbol{u} = \frac{\mathbf{1}}{nK\boldsymbol{v}}, \boldsymbol{v} = \frac{\mathbf{1}}{nK^T\boldsymbol{u}}$$

Automatic Differentiation

Extremely Complex Architectures

Example

LSTM
recurrent
NN



$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Automatic Differentiation

Automatic differentiation:

set of techniques to numerically evaluate the derivative of a function specified by a computer program.

Automatic differentiation is **not**

numerical differentiation

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right) \qquad \frac{\partial f(\mathbf{x})}{\partial x_i} \approx \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$

symbolic differentiation

$$\frac{d}{dx}(f(x) + g(x)) \rightsquigarrow \frac{d}{dx}f(x) + \frac{d}{dx}g(x)$$

$$\frac{d}{dx}(f(x)\,g(x)) \rightsquigarrow \left(\frac{d}{dx}f(x)\right) g(x) + f(x)\left(\frac{d}{dx}g(x)\right) .$$

Automatic Differentiation

$$l_1 = x$$
$$l_n + 1 = 4l_n(1 - l_n)$$

$$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$$

Source: *Automatic differentiation in machine learning, a survey, Baydin et. al, 2015*

Automatic Differentiation

$l_1 = x$
$l_n + 1 = 4l_n(1 - l_n)$

$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$

Manual
Differentiation

$$f'(x) = 128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2) + 64(1-x)(1-2x)^2(1-8x+8x^2)^2 - 64x(1-2x)^2(1-8x+8x^2)^2 - 256x(1-x)(1-2x)(1-8x+8x^2)^2$$

Source: *Automatic differentiation in machine learning, a survey, Baydin et. al, 2015*

Automatic Differentiation

$l_1 = x$
$l_n + 1 = 4l_n(1 - l_n)$

$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$

Manual Differentiation

$f'(x) = 128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2) + 64(1-x)(1-2x)^2(1-8x+8x^2)^2 - 64x(1-2x)^2(1-8x+8x^2)^2 - 256x(1-x)(1-2x)(1-8x+8x^2)^2$

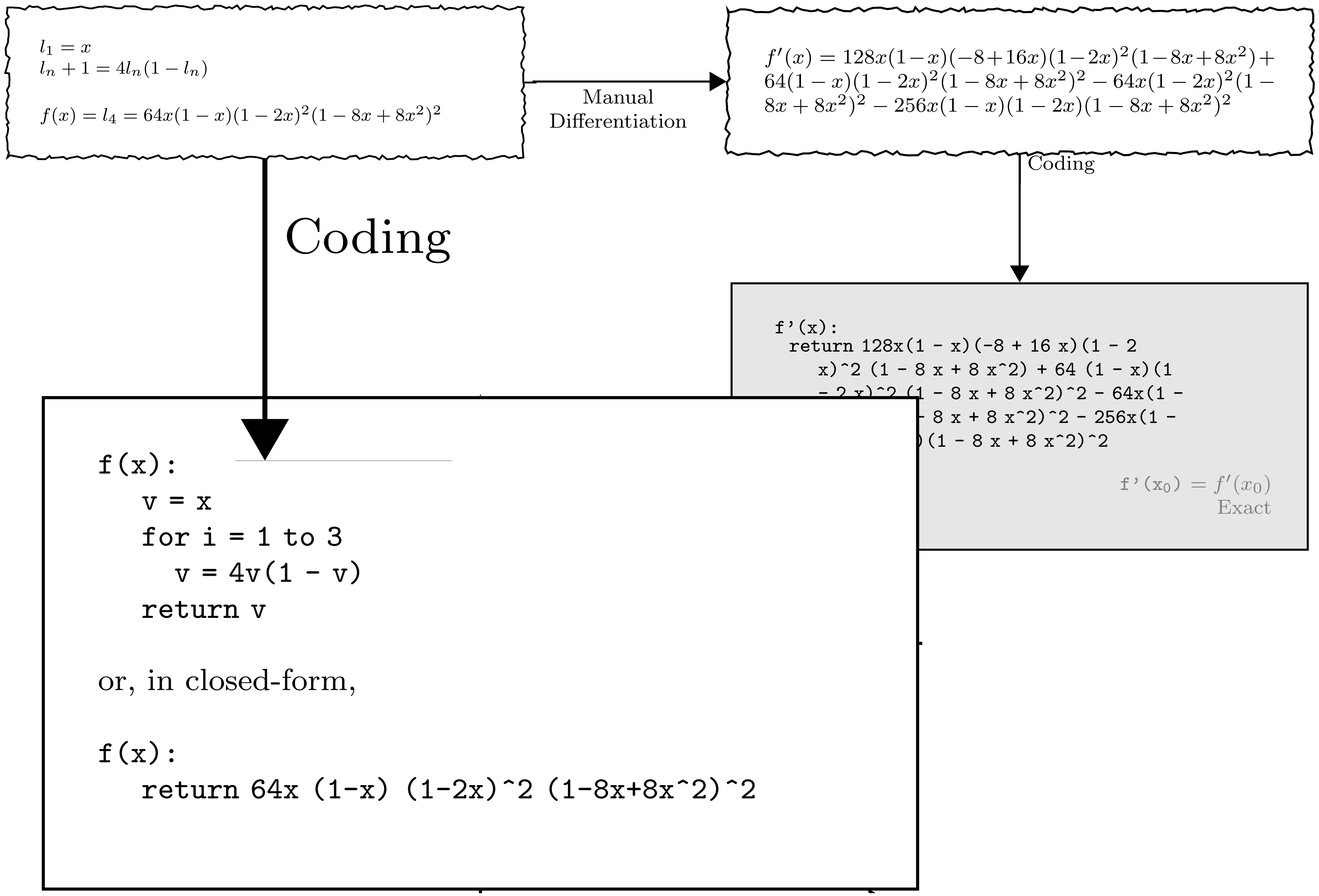
Coding

```
f'(x):
 return 128x(1 - x)(-8 + 16 x)(1 - 2
    x)^2 (1 - 8 x + 8 x^2) + 64 (1 - x)(1
    - 2 x)^2 (1 - 8 x + 8 x^2)^2 - 64x(1 -
    2 x)^2 (1 - 8 x + 8 x^2)^2 - 256x(1 -
    x)(1 - 2 x)(1 - 8 x + 8 x^2)^2
```

f'(x_0) $= f'(x_0)$
Exact

Source: *Automatic differentiation in machine learning, a survey, Baydin et. al, 2015*

Automatic Differentiation



Source: *Automatic differentiation in machine learning, a survey, Baydin et. al, 2015*

Automatic Differentiation

$l_1 = x$
$l_n + 1 = 4l_n(1 - l_n)$

$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$

Manual Differentiation

$f'(x) = 128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2) + 64(1-x)(1-2x)^2(1-8x+8x^2)^2 - 64x(1-2x)^2(1-8x+8x^2)^2 - 256x(1-x)(1-2x)(1-8x+8x^2)^2$

Coding

Coding

```
f(x):
  v = x
  for i = 1 to 3
    v = 4v(1 - v)
  return v
```

or, in closed-form,

```
f(x):
  return 64x (1-x) (1-2x)^2 (1-8x+8x^2)^2
```

Symbolic Differentiation of the Closed-form

```
f'(x):
 return 128x(1 - x)(-8 + 16 x)(1 - 2
    x)^2 (1 - 8 x + 8 x^2) + 64 (1 - x)(1
    - 2 x)^2 (1 - 8 x + 8 x^2)^2 - 64x(1 -
    2 x)^2 (1 - 8 x + 8 x^2)^2 - 256x(1 -
    x)(1 - 2 x)(1 - 8 x + 8 x^2)^2
```

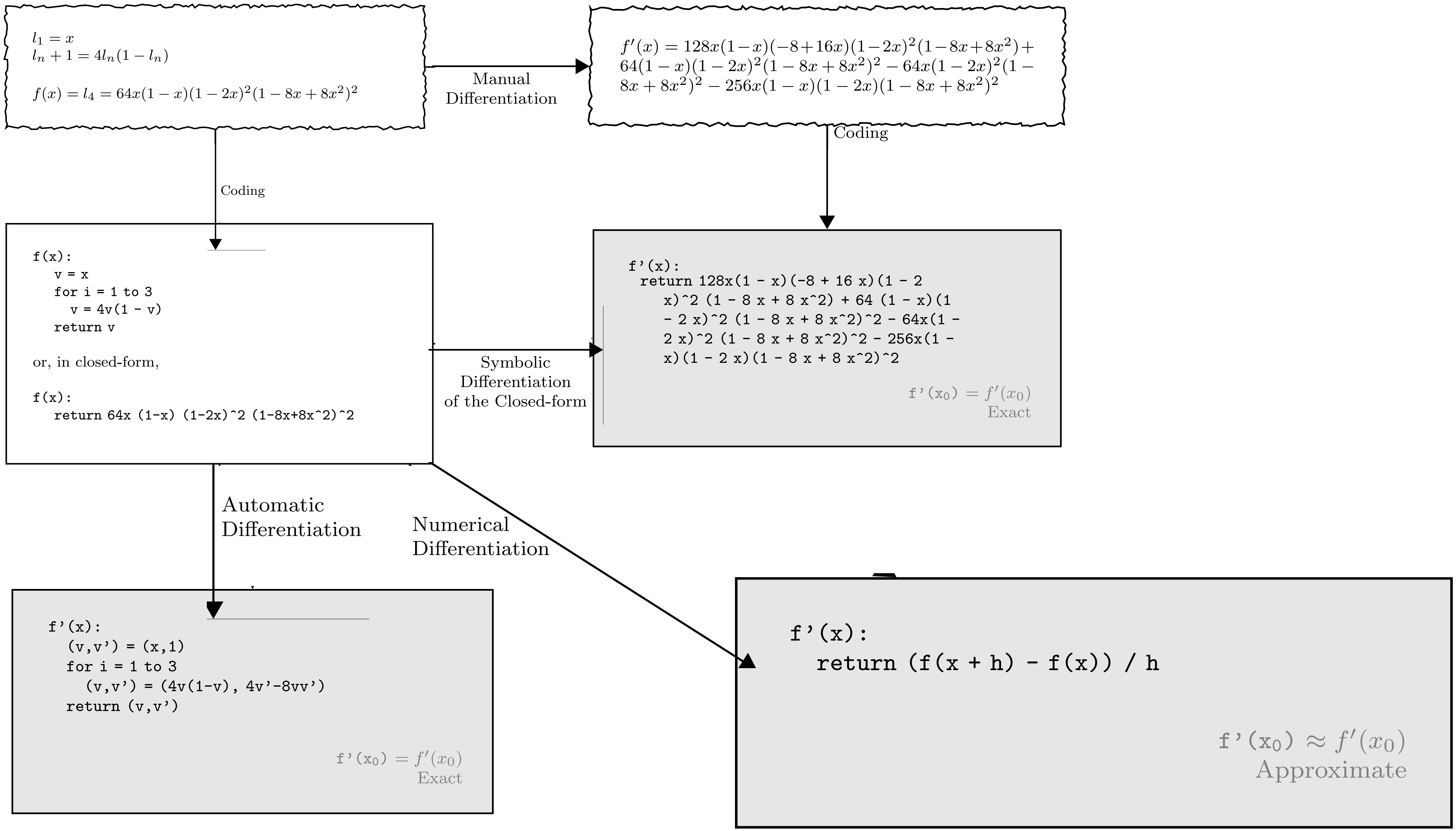
f'(x_0) $= f'(x_0)$
Exact

Automatic Differentiation

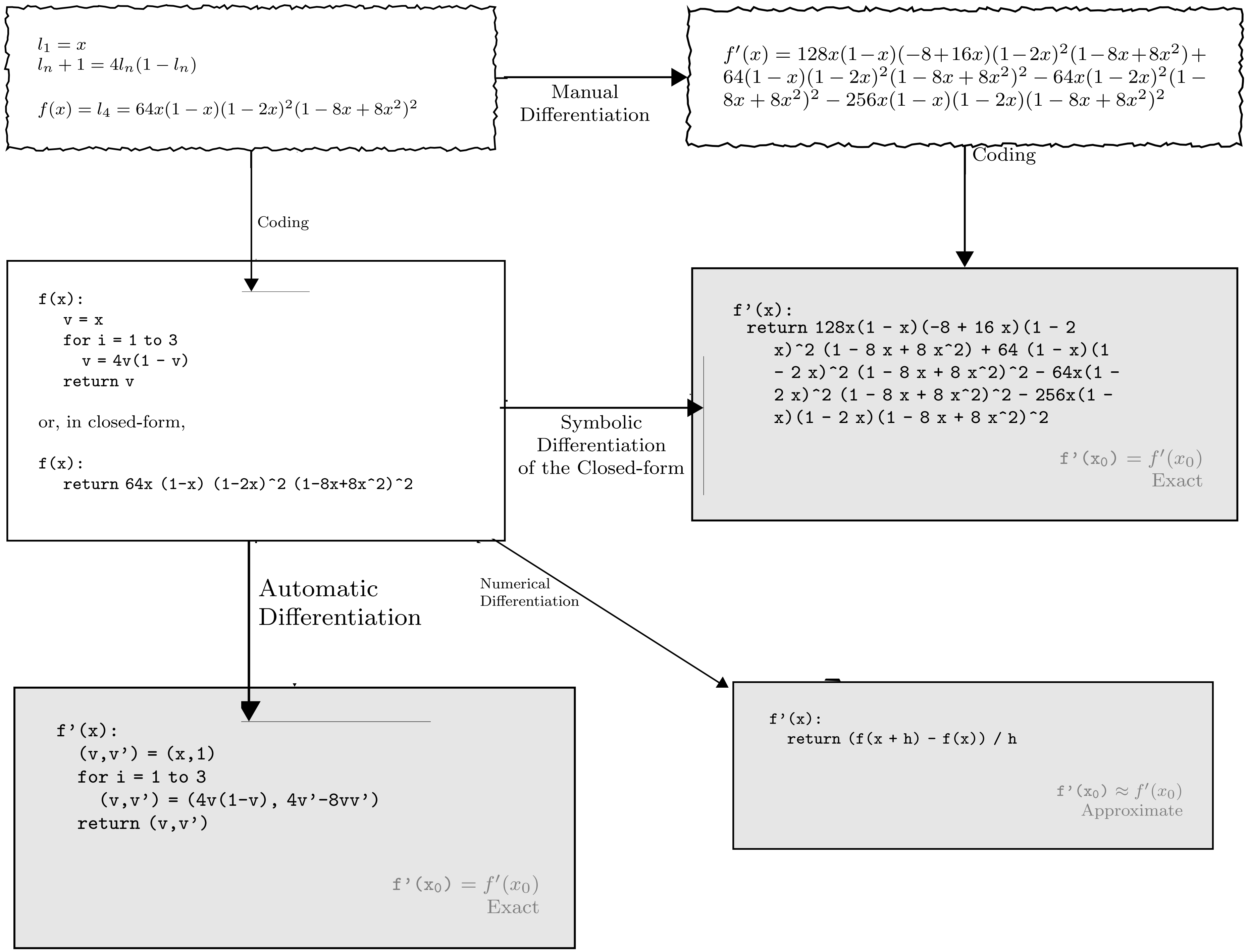
```
f'(x):
  (v,v') = (x,1)
  for i = 1 to 3
    (v,v') = (4v(1-v), 4v'-8vv')
  return (v,v')
```

f'(x_0) $= f'(x_0)$
Exact

Automatic Differentiation



Automatic Differentiation



Automatic Differentiation

n	l_n	$\frac{d}{dx} l_n$	$\frac{d}{dx} l_n$ (Optimized)
1	x	1	1
2	$4x(1-x)$	$4(1-x) - 4x$	$4 - 8x$
3	$16x(1-x)(1-2x)^2$	$16(1-x)(1-2x)^2 - 16x(1-2x)^2 - 64x(1-x)(1-2x)$	$16(1 - 10x + 24x^2 - 16x^3)$
4	$64x(1-x)(1-2x)^2(1-8x+8x^2)^2$	$128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2) + 64(1-x)(1-2x)^2(1-8x+8x^2)^2 - 64x(1-2x)^2(1-8x+8x^2)^2 - 256x(1-x)(1-2x)(1-8x+8x^2)^2$	$64(1-42x+504x^2-2640x^3+7040x^4-9984x^5+7168x^6-2048x^7)$

Automatic Differentiation

Computer code for $f(x_1, x_2) = x_1 x_2 + \sin(x_1)$ might read

Original program

$$
\begin{aligned}
w_1 &= x_1\\
w_2 &= x_2\\
w_3 &= w_1 w_2\\
w_4 &= \sin(w_1)\\
w_5 &= w_3 + w_4
\end{aligned}
$$

Dual program

$$
\begin{aligned}
\dot{w}_1 &= 0\\
\dot{w}_2 &= 1\\
\dot{w}_3 &= \dot{w}_1 w_2 + w_1 \dot{w}_2 = 0 \cdot x_2 + x_1 \cdot 1 = x_1\\
\dot{w}_4 &= \cos(w_1)\dot{w}_1 = \cos(x_1) \cdot 0 = 0\\
\dot{w}_5 &= \dot{w}_3 + \dot{w}_4 = x_1 + 0 = x_1
\end{aligned}
$$

and

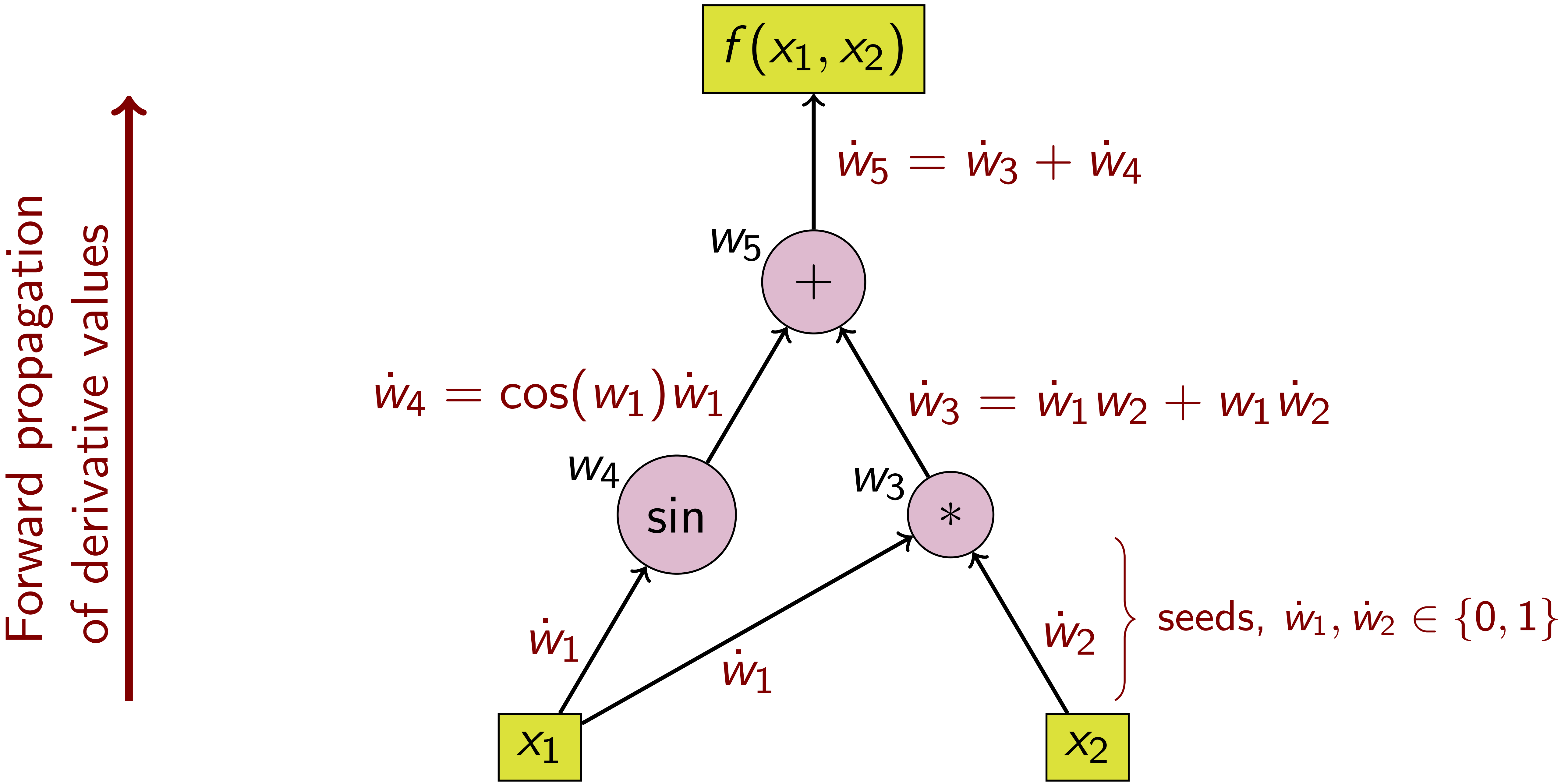
$$\frac{\partial f}{\partial x_2} = x_1$$

The chain rule

$$\frac{\partial f}{\partial x_2} = \frac{\partial f}{\partial w_5} \frac{\partial w_5}{\partial w_3} \frac{\partial w_3}{\partial w_2} \frac{\partial w_2}{\partial x_2}$$

ensures that we can *propagate* the dual components throughout the computation.

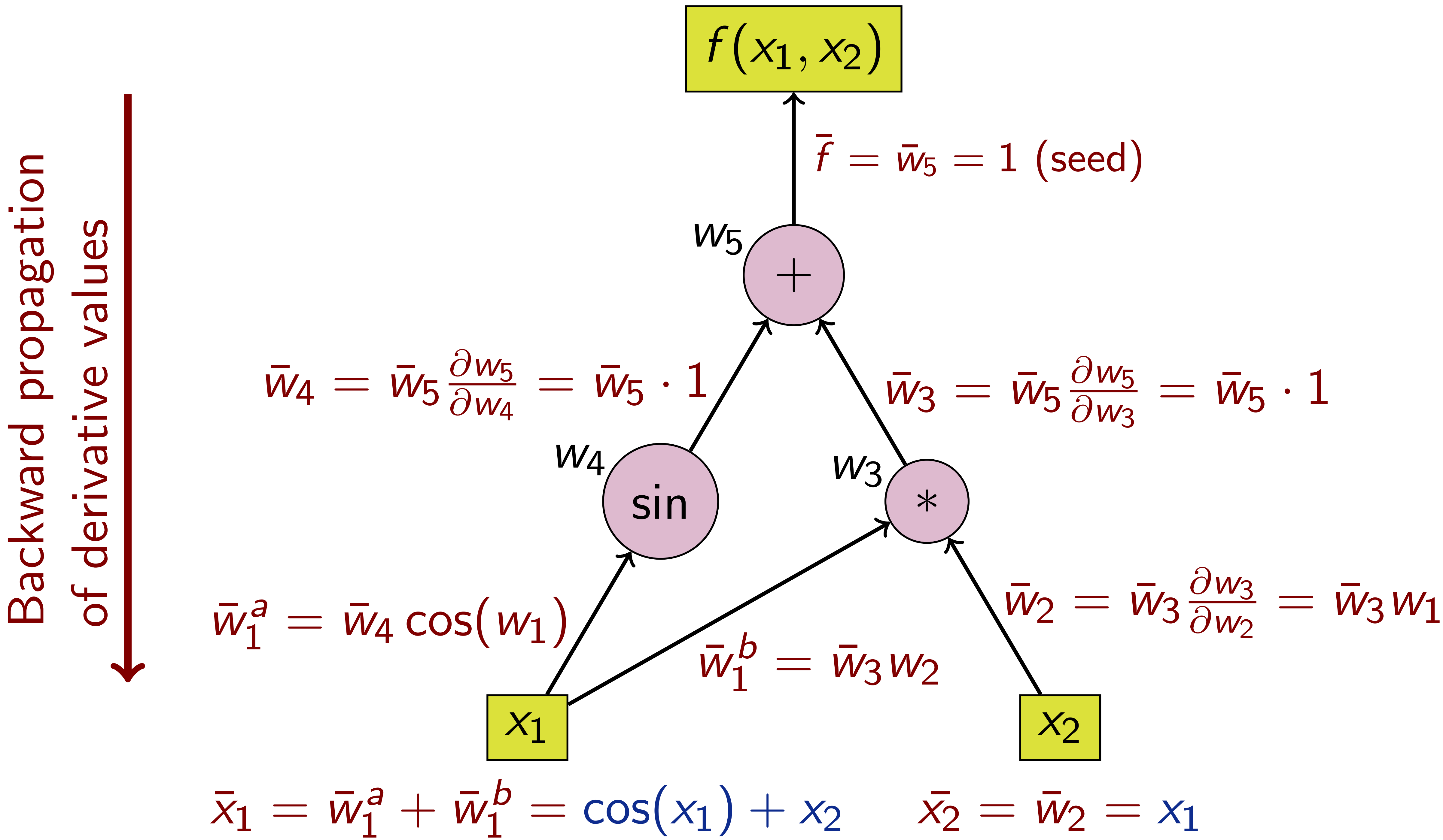
Automatic Differentiation



$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \frac{\partial y}{\partial w_1}\left(\frac{\partial w_1}{\partial w_2}\frac{\partial w_2}{\partial x}\right) = \frac{\partial y}{\partial w_1}\left(\frac{\partial w_1}{\partial w_2}\left(\frac{\partial w_2}{\partial w_3}\frac{\partial w_3}{\partial x}\right)\right) = \cdots$$

Source: Havard Berland, NTNU

Automatic Differentiation



$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \left(\frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \left(\left(\frac{\partial y}{\partial w_3}\frac{\partial w_3}{\partial w_2}\right)\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \cdots$$

Source: Havard Berland, NTNU

Automatic Differentiation

Given $F\colon \mathbf{R}^n \mapsto \mathbf{R}^m$ and the Jacobian $J = DF(\mathbf{x}) \in \mathbf{R}^{m\times n}$.

$$J = DF(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

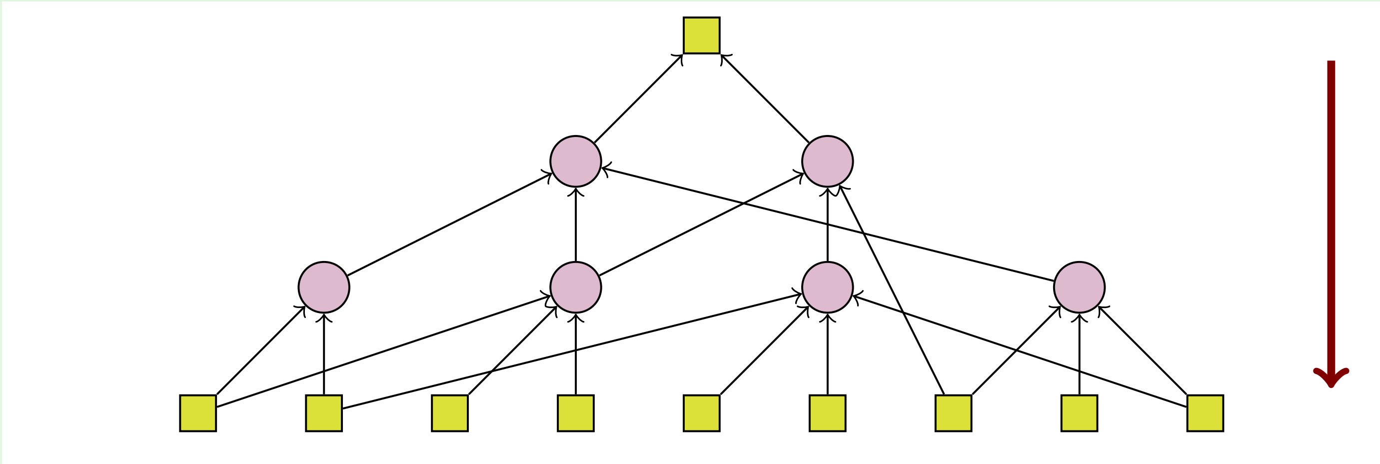
Backward sweep can compute one column

Forward sweep can compute one column

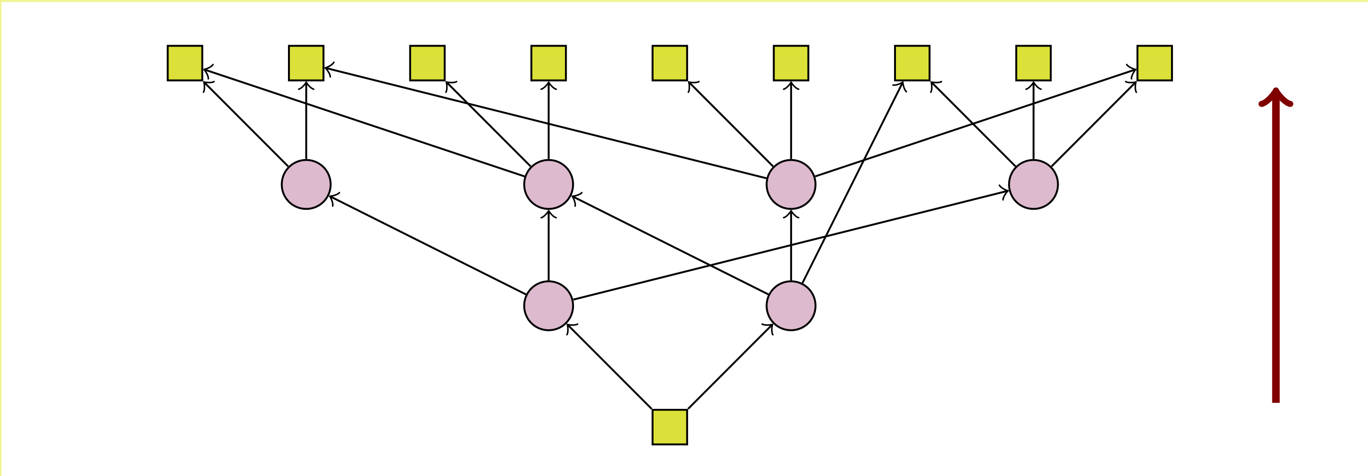
Source: Havard Berland, NTNU

Automatic Differentiation

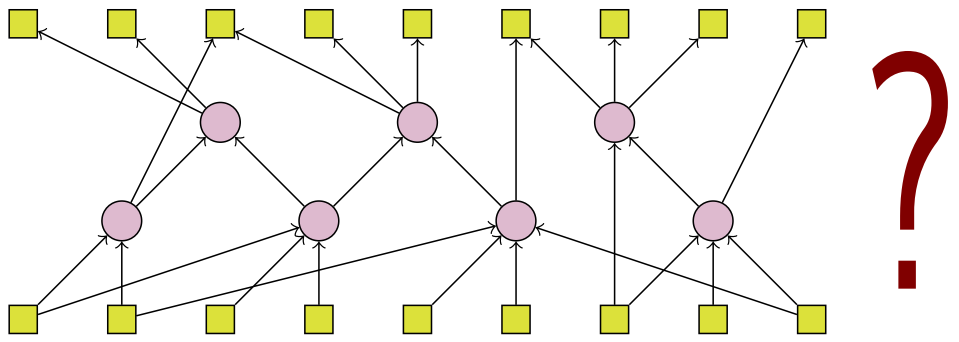
Reverse mode suitable for $F : \mathbb{R}^p \mapsto \mathbb{R}$



Forward mode suitable for $F : \mathbb{R} \mapsto \mathbb{R}^p$



$$F : \mathbb{R}^d \mapsto \mathbb{R}^p \ ?$$



Source: Havard Berland, NTNU