Transprecision Techniques for Linear Solvers and Non-Linear Regressions

JunKyu Lee

Ax=b Entrans
PRECOMP Open Transprecision Computing
QUEEN'S UNIVERSITY BELFAST
ECIT THE INSTITUTE OF ELECTRONICS COMMUNICATIONS AND INFORMATION TECHNOLOGY

Contents

Introduction. Transprecision Computing Concepts (15 mins)

Part A. Transprecision techniques for Linear Solver (30 mins)

5 mins break;

Part B. Transprecision techniques for Kernel Recursive Least Squares (30 mins)

10 mins QnA

Contents

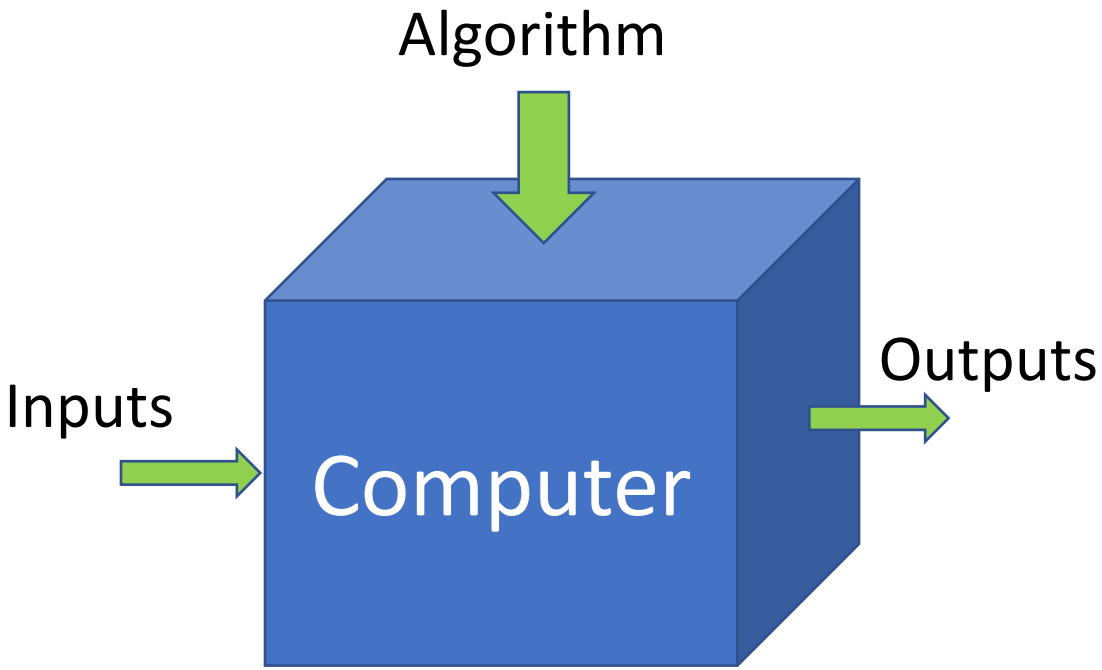
Beside transprecision skills, we will also learn machine learning and linear algebra such as:

- **Accuracy and Precision**
- **Gaussian Elimination (LU Decomposition)**
- **LU direct Solver**
- **Iterative Refinement**
- **Condition Number**
- **Global Positioning System**
- **Kernel Method Machine Learning**

Transprecision Concept

Computation Requirements

Precision vs Accuracy?

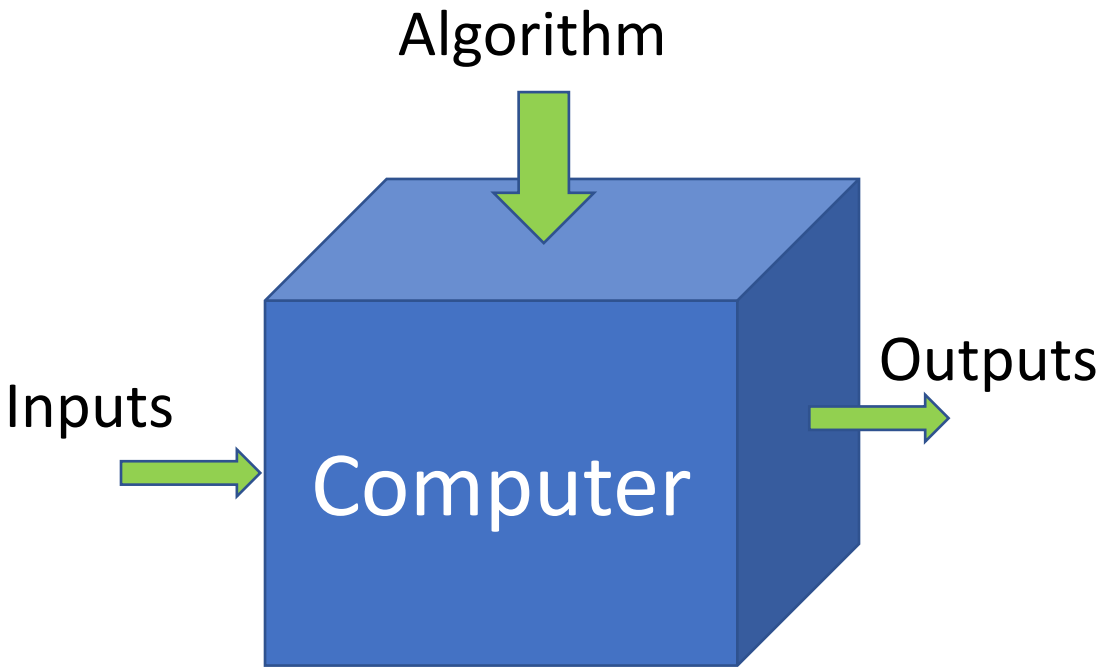


Computation Requirements

Precision vs Accuracy:

Accuracy: the absolute or relative error of an approximate quantity.

Precision: the accuracy with which the basic arithmetic operations +,−, ∗, / are performed. - “Accuracy and Stability of Numerical Algorithms”, NJ Higham -

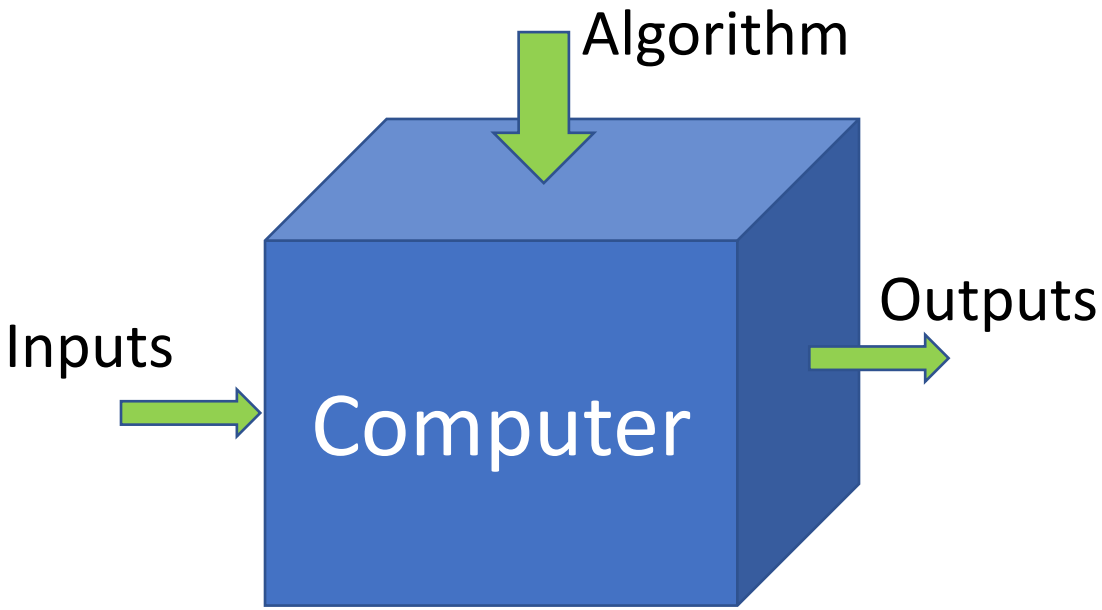


Computation Requirements

Precision vs Accuracy:

Accuracy: the absolute or relative error of an approximate quantity.

Precision: Every basic operation accuracy should be within one machine epsilon (IEEE 754). Single Precision (32bits) Machine Epsilon: 2^{-24} Double Precision (64bits) Machine Epsilon: 2^{-53}

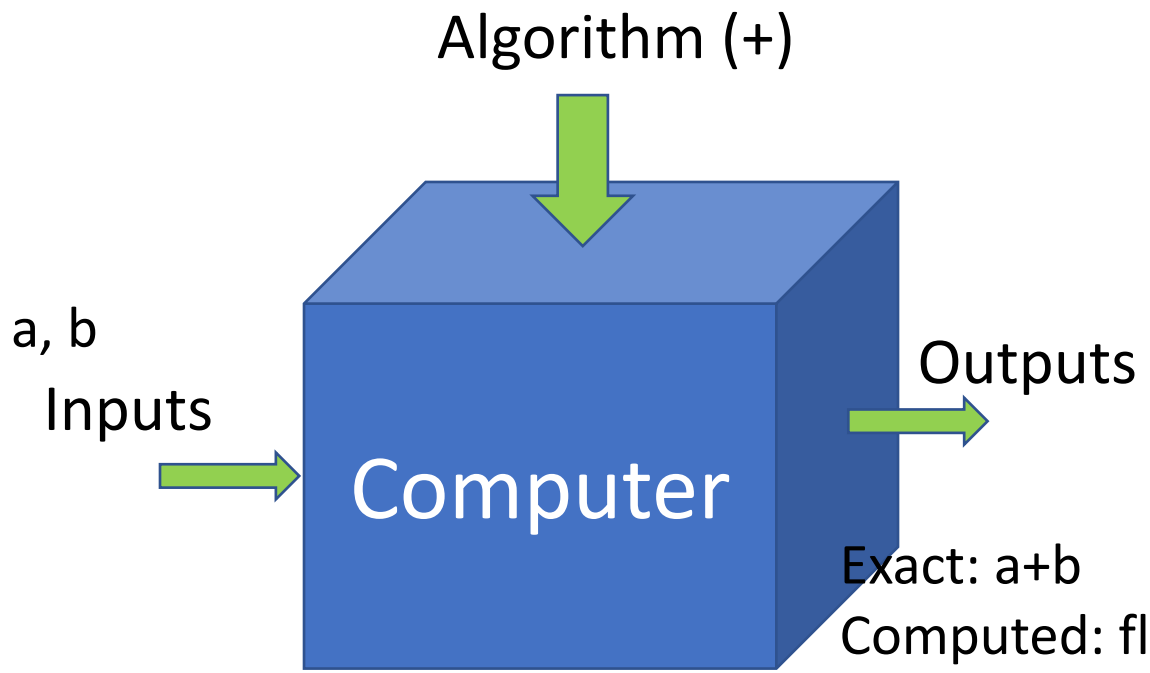


Computation Requirements

Eg. Accuracy and Precision for single precision arith Linear Solver A x = b

Accuracy: $||x^* - x||/||x^*||$

Precision: machine epsilon for single precision = 2^{-24}.



Exact: a+b

Computed: fl(a+b)

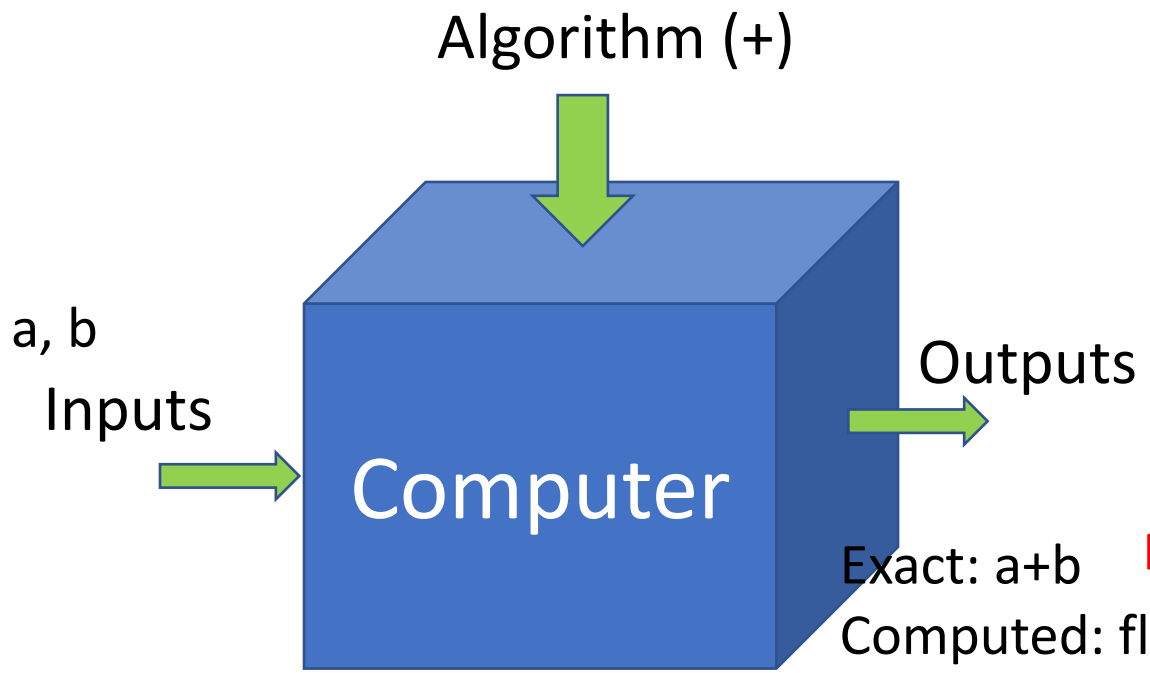
$a + b - 2^{-24} \leq fl(a+b) < a + b + 2^{-}$

Computation Requirements

Eg. Accuracy and Precision for single precision arith Linear Solver A x = b

Accuracy: $||x^* - x||/||x^*||$

Precision: machine epsilon for single precision = 2^{-24}.



Exact: a+b

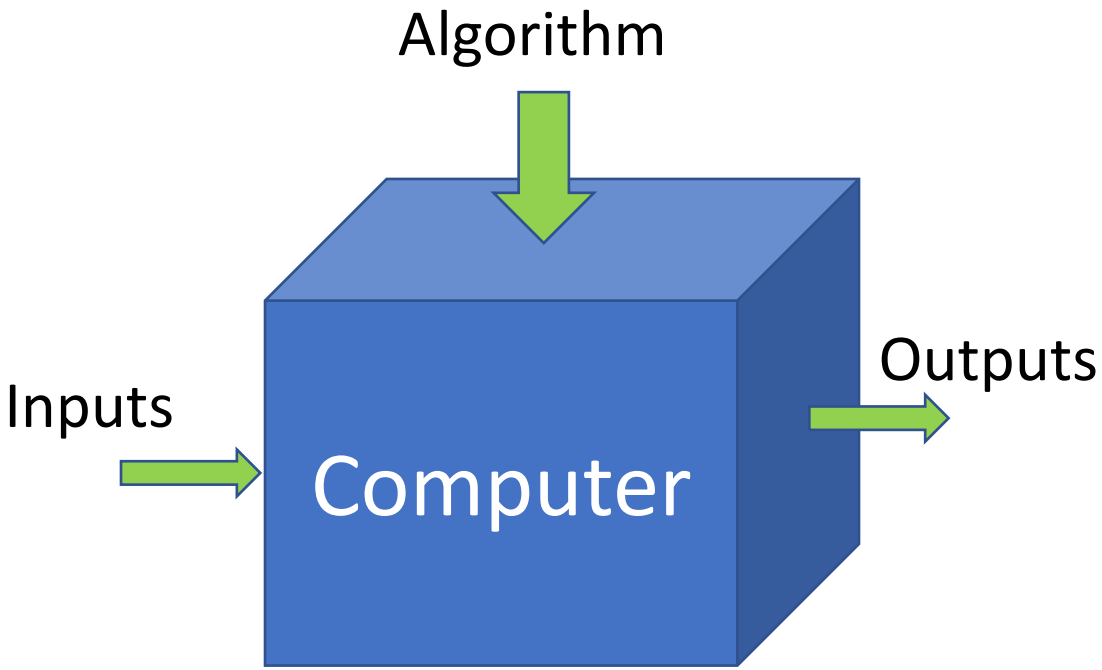
Round-off errors

Computed: fl(a+b)

$a + b - 2^{-24} \leq fl(a+b) < a + b + 2^{-}$

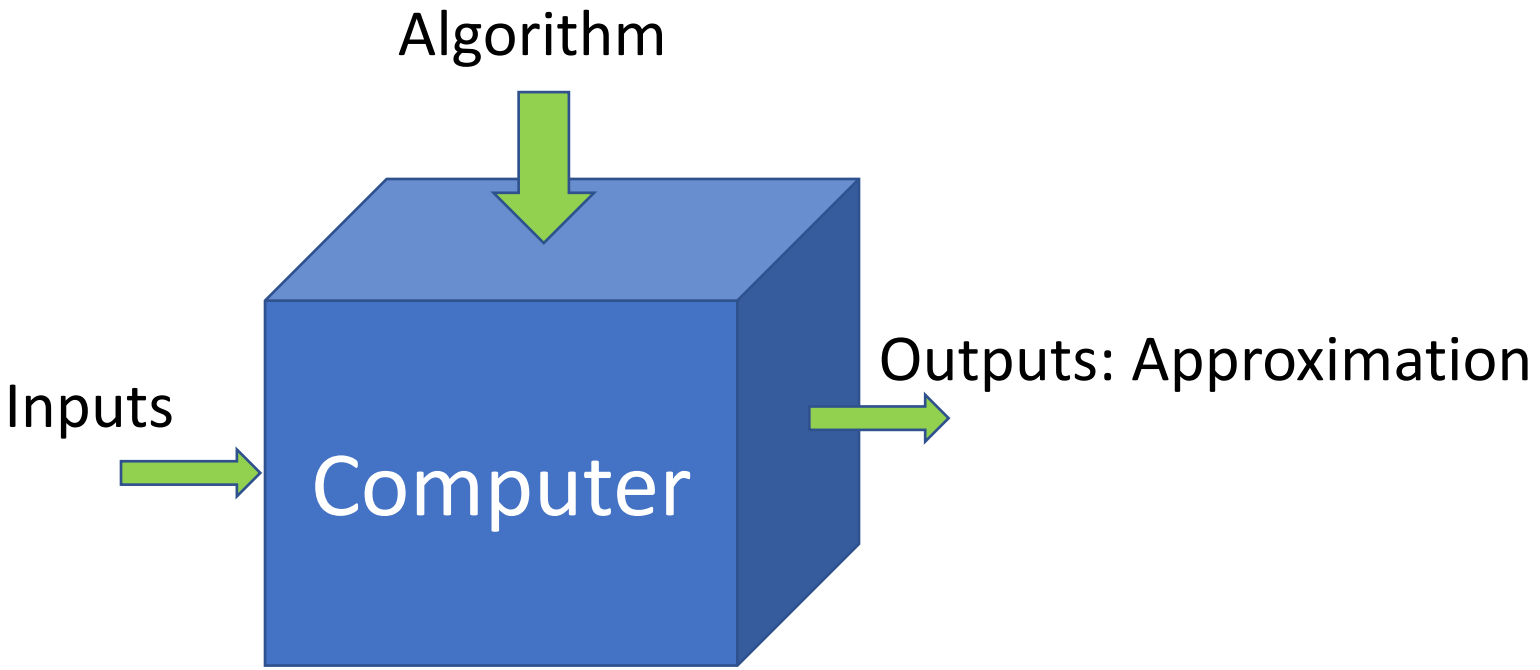
Computation Requirements

Finite Precision Arithmetic vs Exact Arithmetic ?



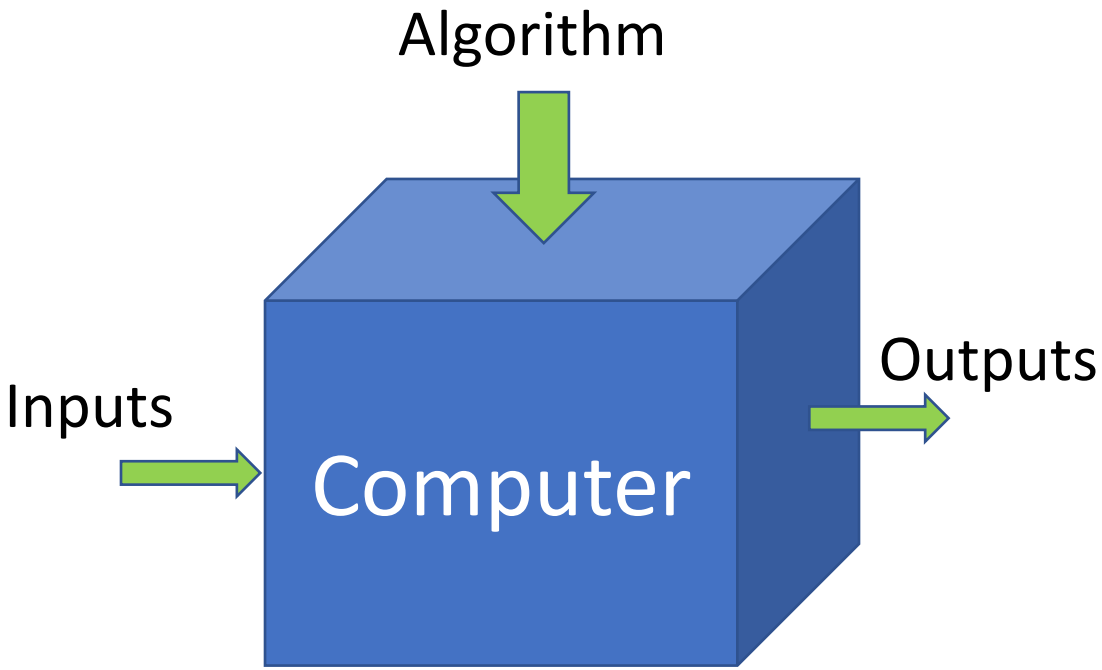
Computation Requirements

Finite Precision Arithmetic vs Exact Arithmetic ?



Computation Requirements

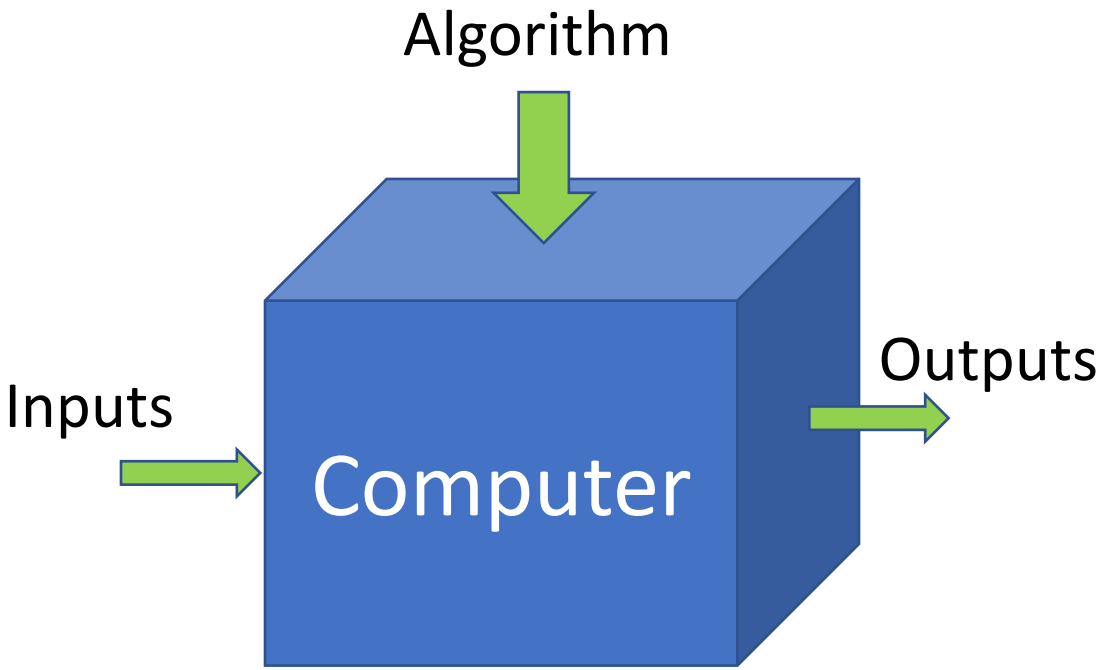
Solution Accuracy ?



Computation Requirements

If Solution Accuracy is Good enough,

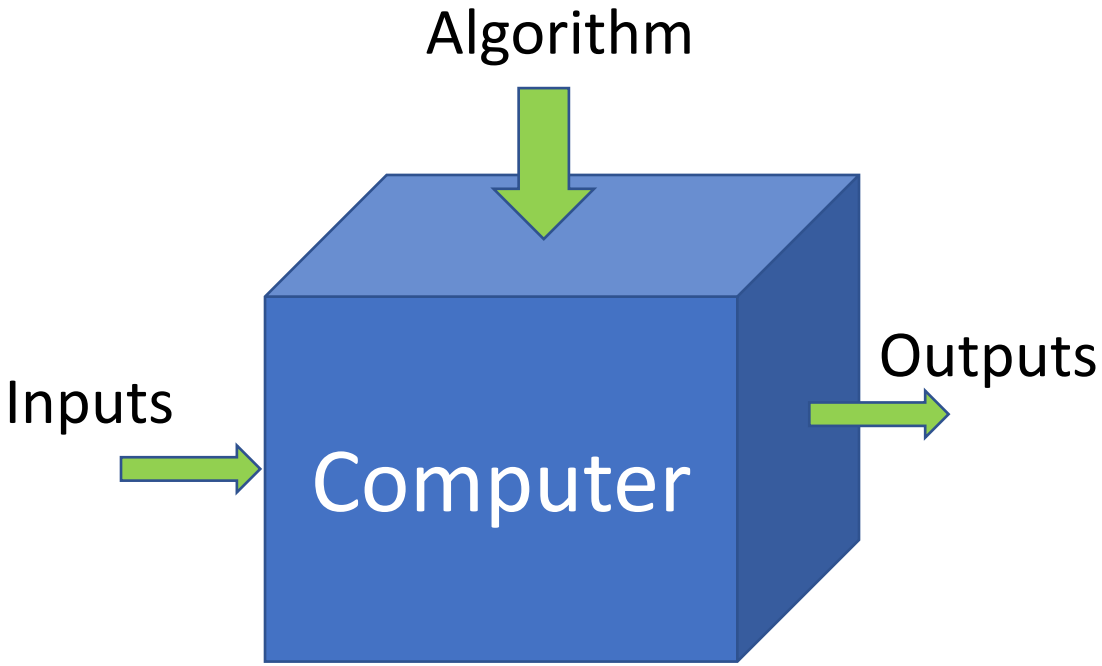
Speed and Energy?



Computation Requirements

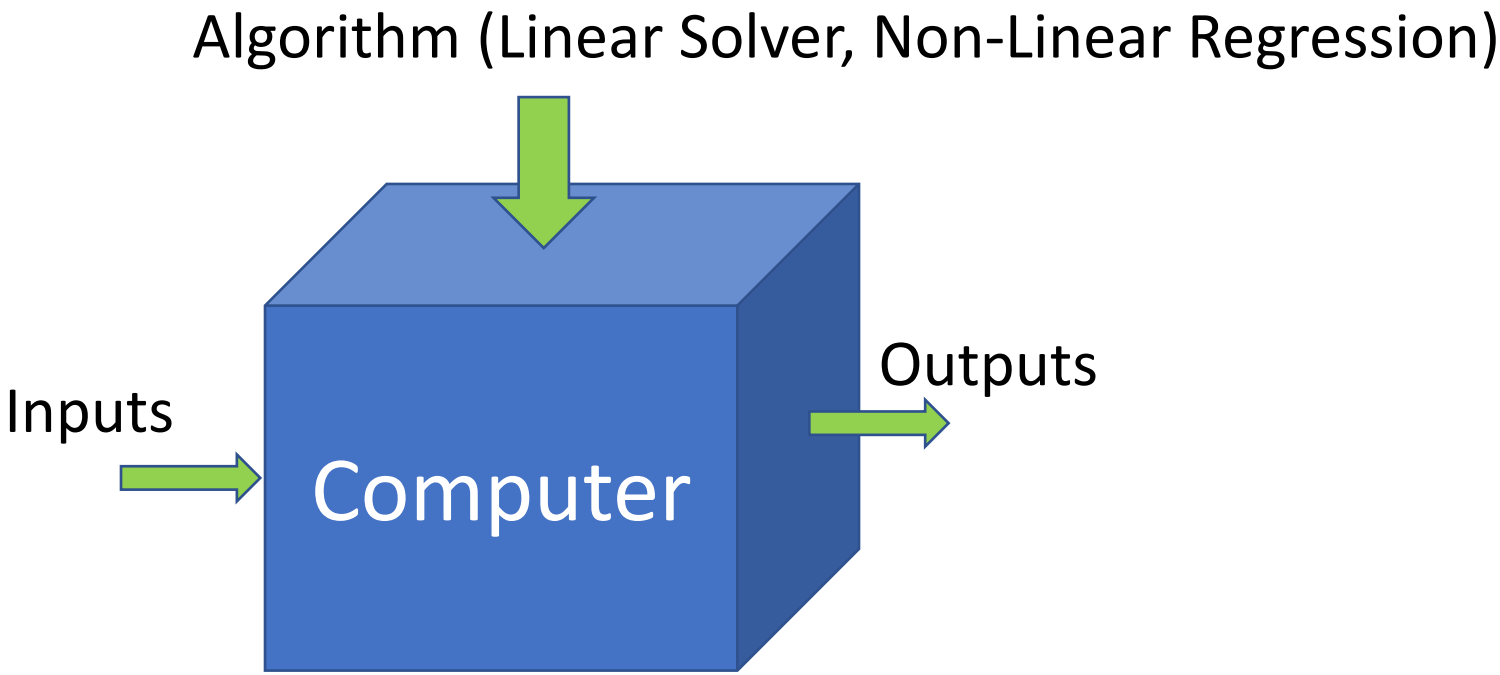
If Solution Accuracy is Good enough,

Transprecision Techniques minimizes Energy and Execution time without losing Solution Accuracy!



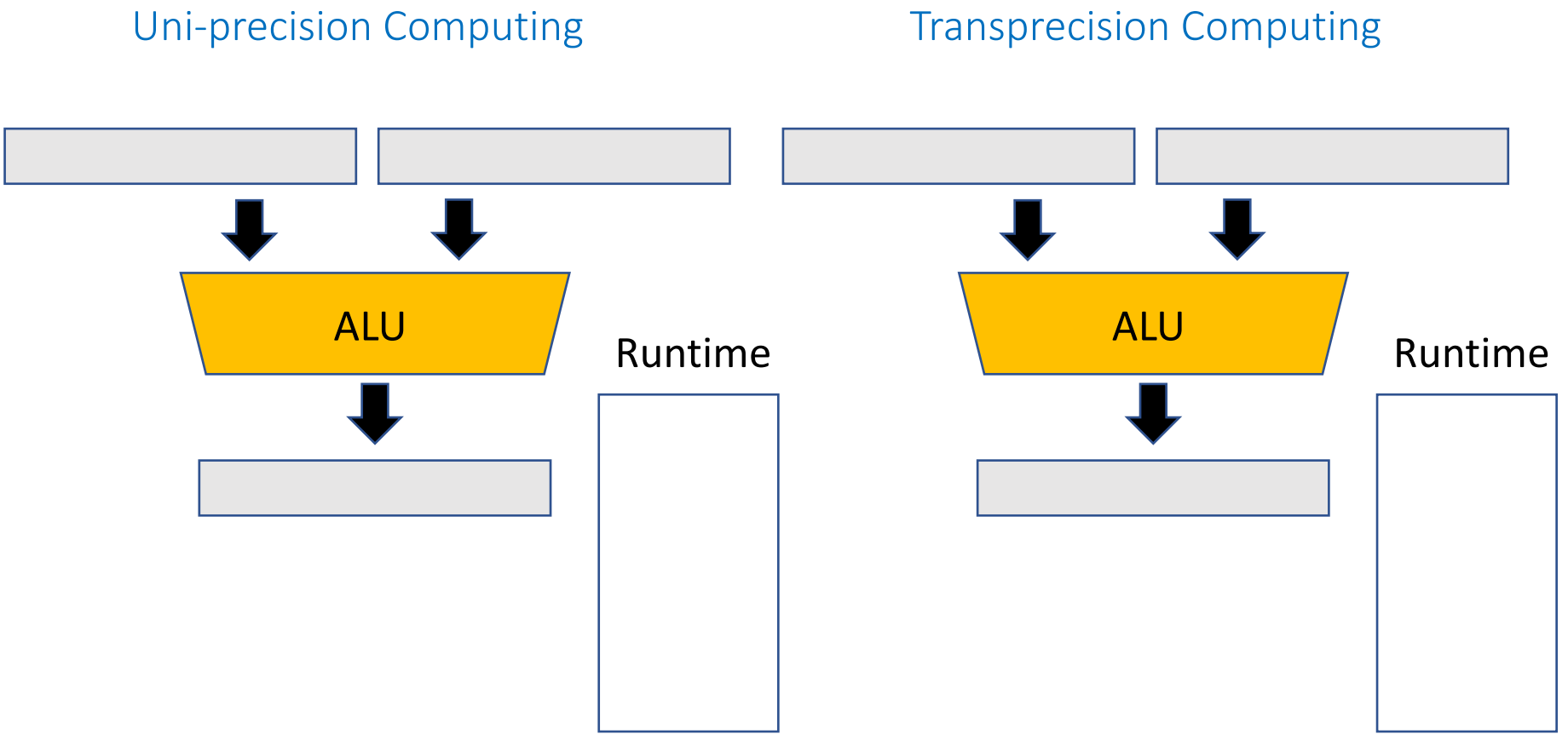
Computation Requirements

How?



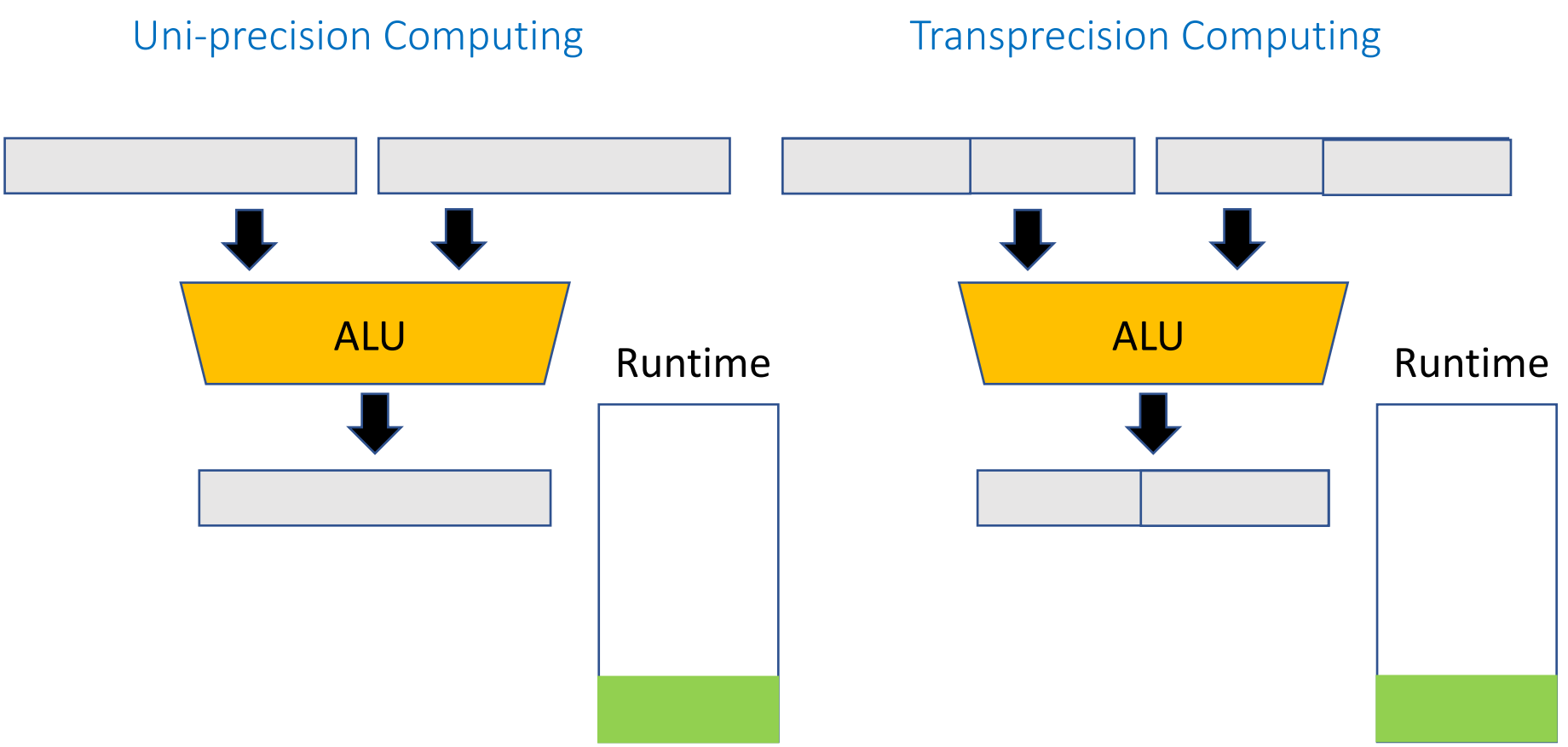
Transprecision Computing Simple Demo

Execution time for 7 operations on CPU/GPU with double precision data



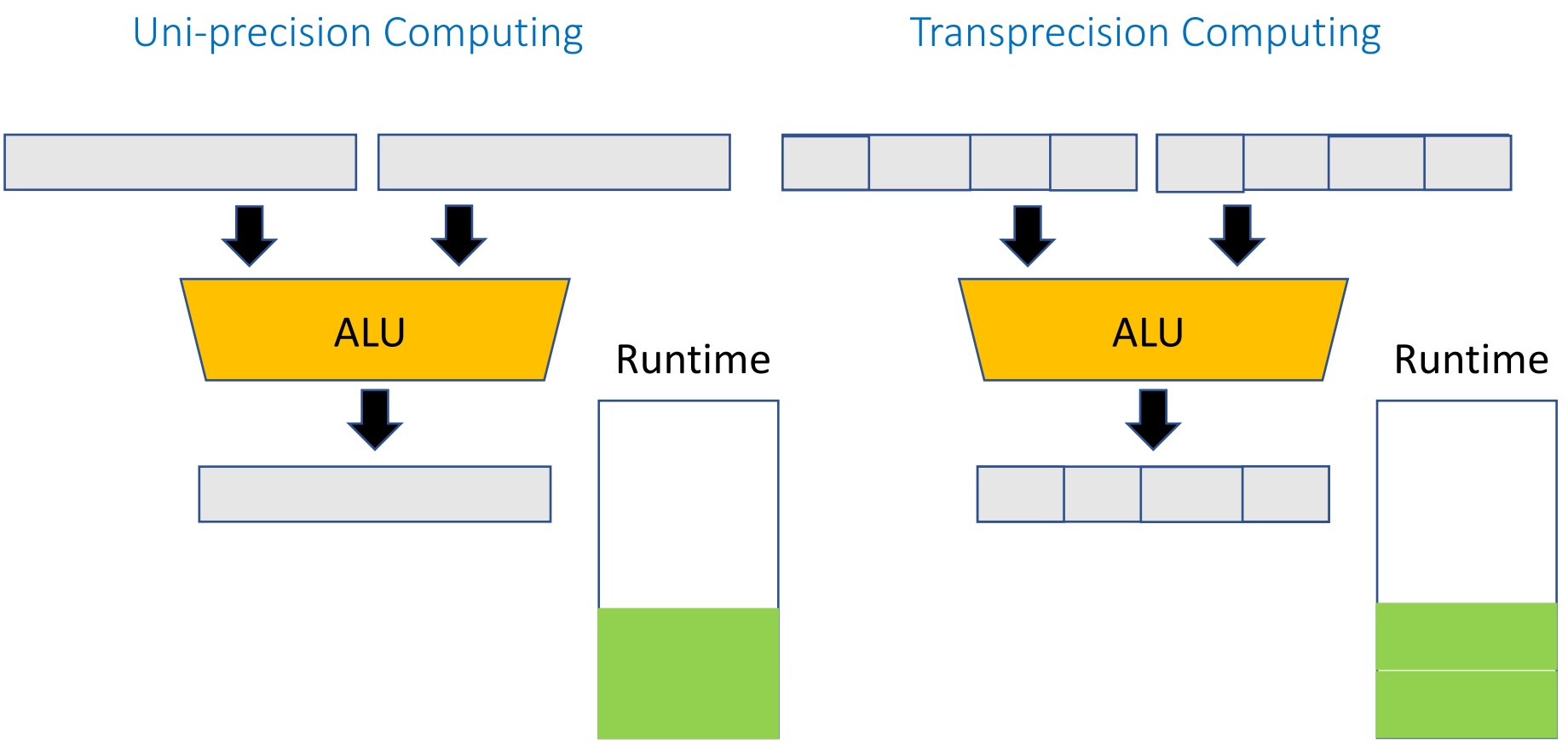
Transprecision Computing Simple Demo

Execution time for 7 operations on CPU/GPU with double precision data



Transprecision Computing Simple Demo

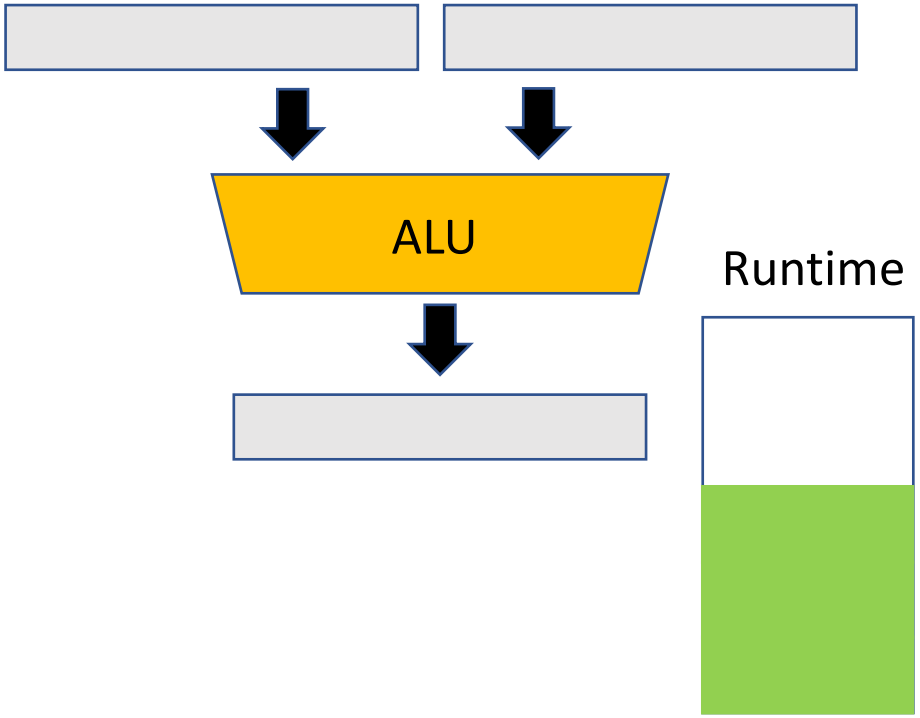
Execution time for 7 operations on CPU/GPU with double precision data



Transprecision Computing Simple Demo

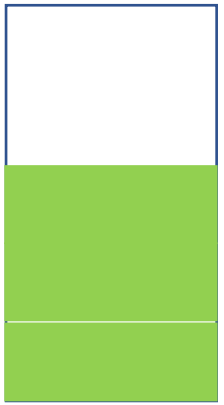
Execution time for 7 operations on CPU/GPU with double precision data

Uni-precision Computing



Transprecision Computing

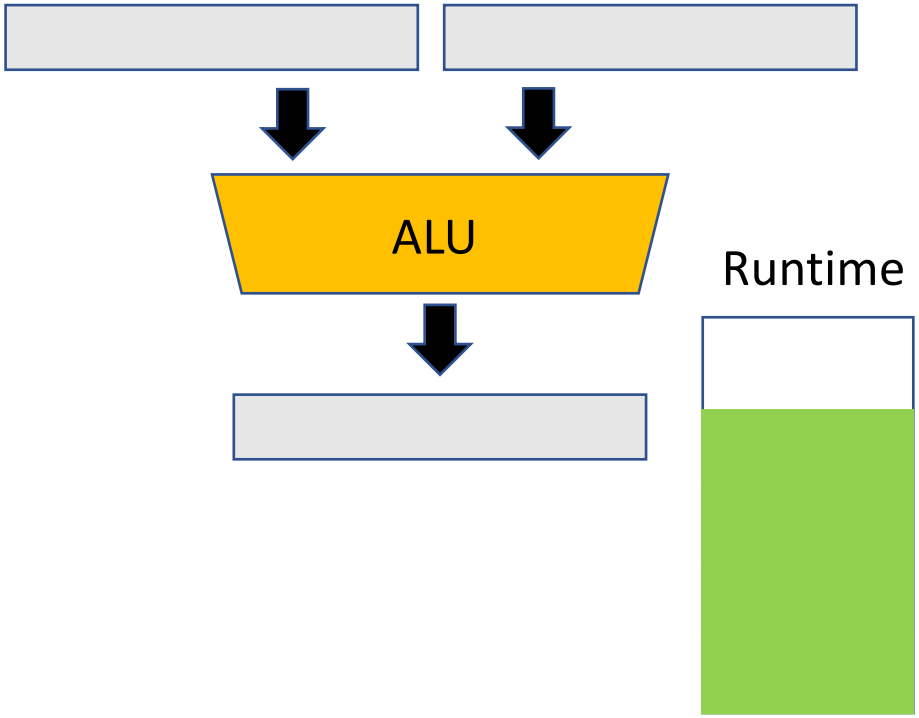
DONE!



Transprecision Computing Simple Demo

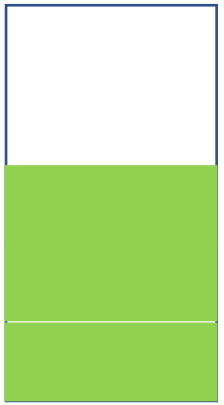
Execution time for 7 operations on CPU/GPU with double precision data

Uni-precision Computing



Transprecision Computing

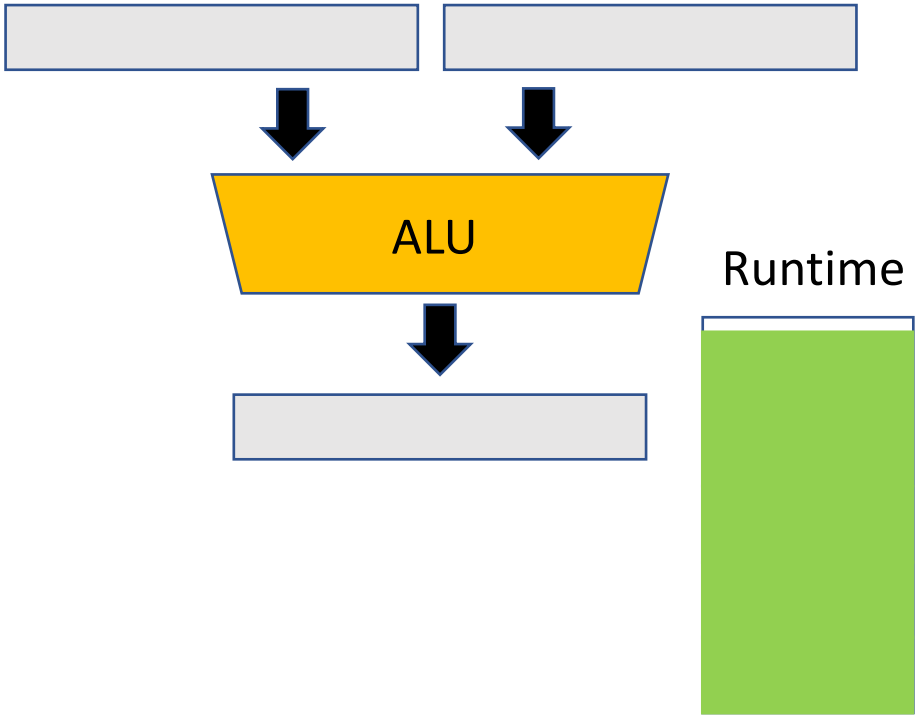
DONE!



Transprecision Computing Simple Demo

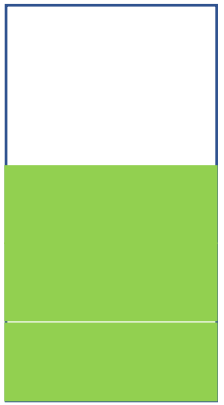
Execution time for 7 operations on CPU/GPU with double precision data

Uni-precision Computing



Transprecision Computing

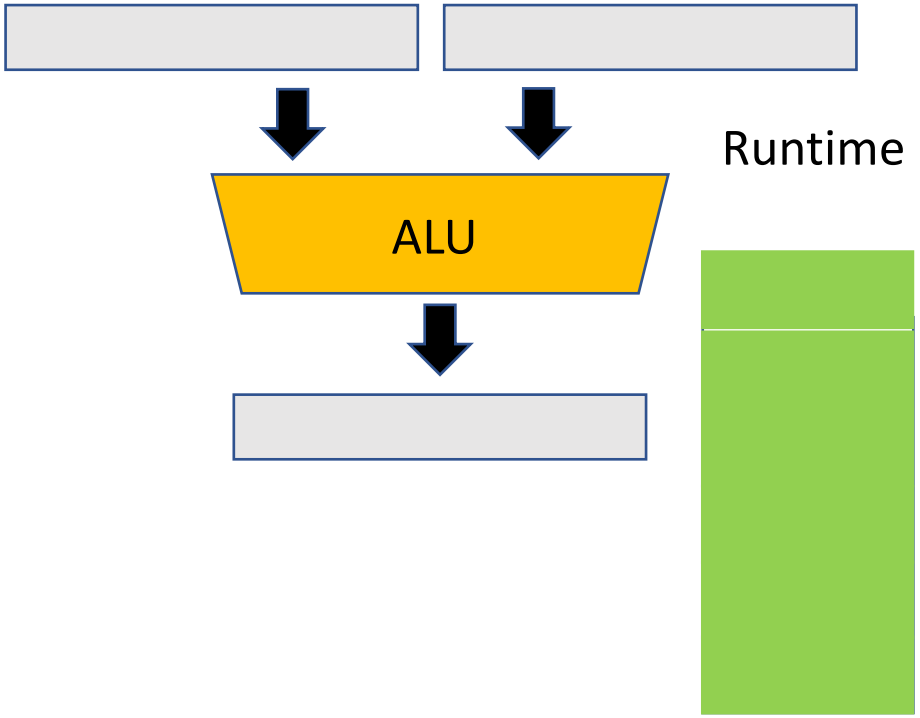
DONE!



Transprecision Computing Simple Demo

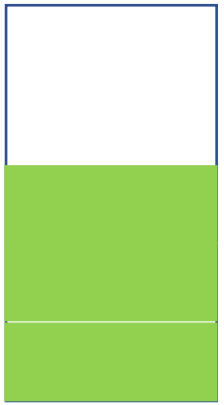
Execution time for 7 operations on CPU/GPU with double precision data

Uni-precision Computing



Transprecision Computing

DONE!



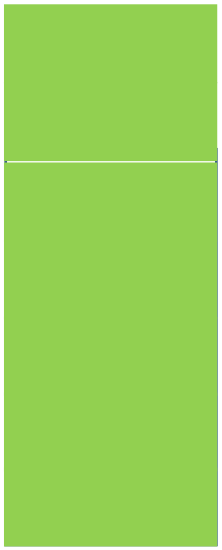
Transprecision Computing Simple Demo

Execution time for 7 operations on CPU/GPU with double precision data

Uni-precision Computing

DONE!

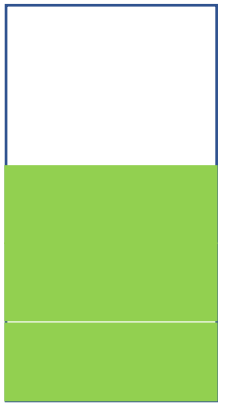
Runtime



Transprecision Computing

DONE!

Runtime

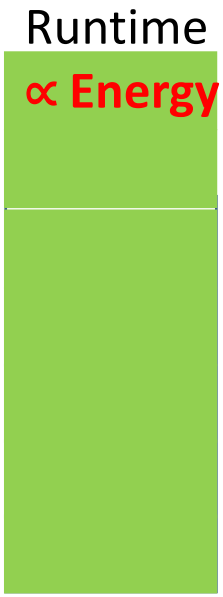


Transprecision Computing Simple Demo

Execution time for 7 operations on CPU/GPU with double precision data

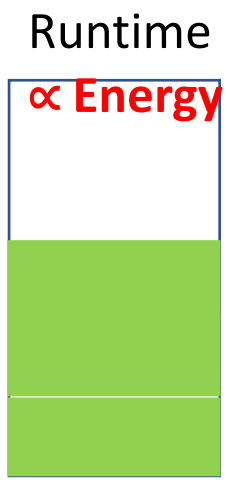
Uni-precision Computing

DONE!



Transprecision Computing

DONE!



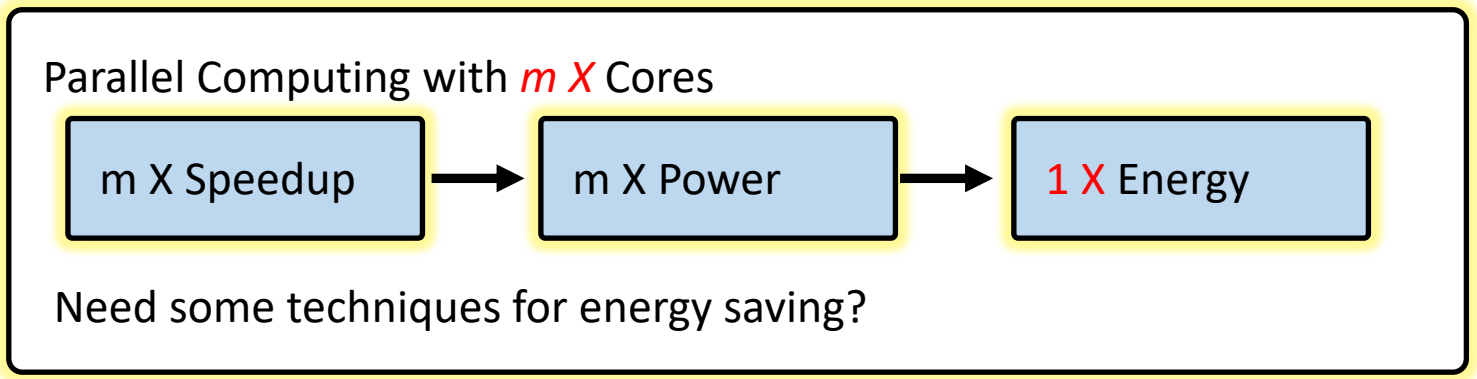
Transprecision Computing Simple Demo

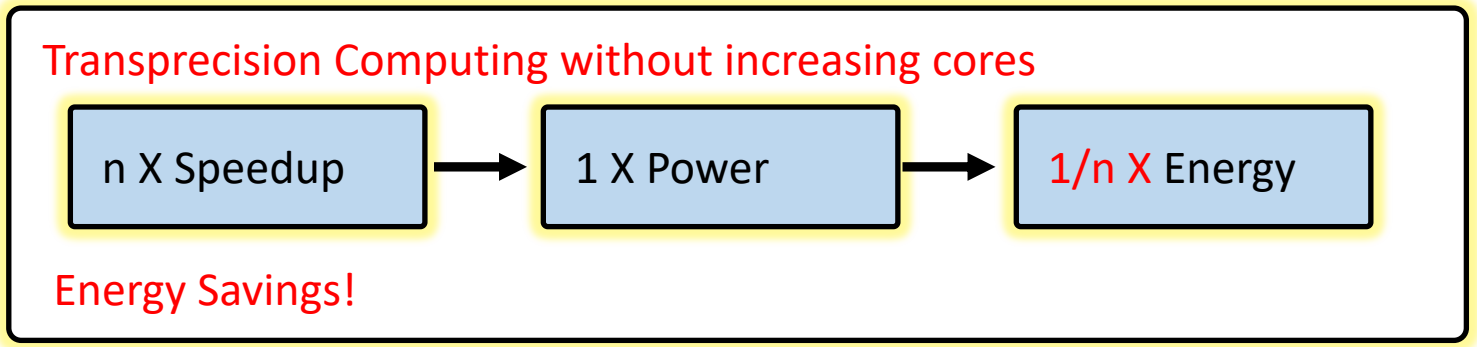
Transprecision Techniques seek

minimised precision arithmetic without losing solution accuracy

Transprecision Computing vs Parallel Computing

Reduced runtime without increasing number of cores
=> Direct Energy Savings!





Transprecision Impact on GPUs/FPGAs

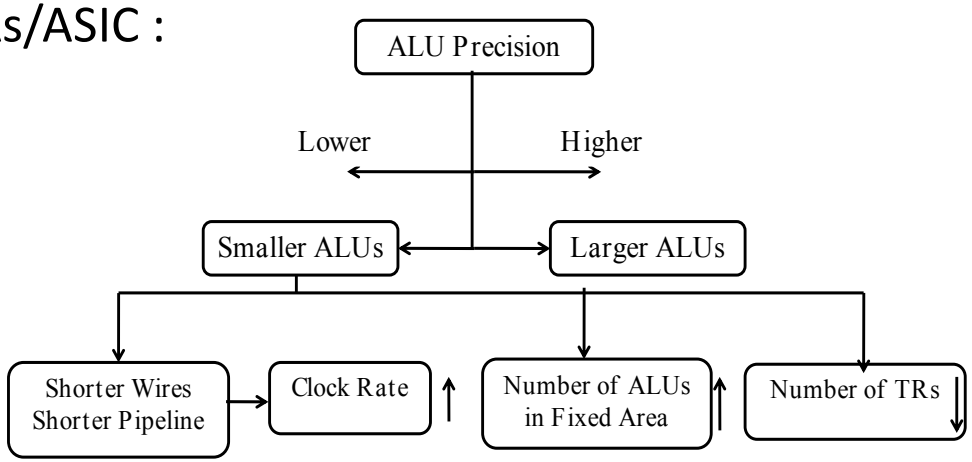
NVidia Pascal GPU (P100) with NVLink:

Half precision: 21.2 TeraFlops
Single precision: 10.6 TeraFlops
Double precision: 5.3 TeraFlops

NVidia Volta GPU (V100) with NVLink:

Half precision (Tensor Core): 125 TeraFlops
Single precision: 15.7 TeraFlops
Double precision: 7.8 TeraFlops

FPGAs/ASIC :

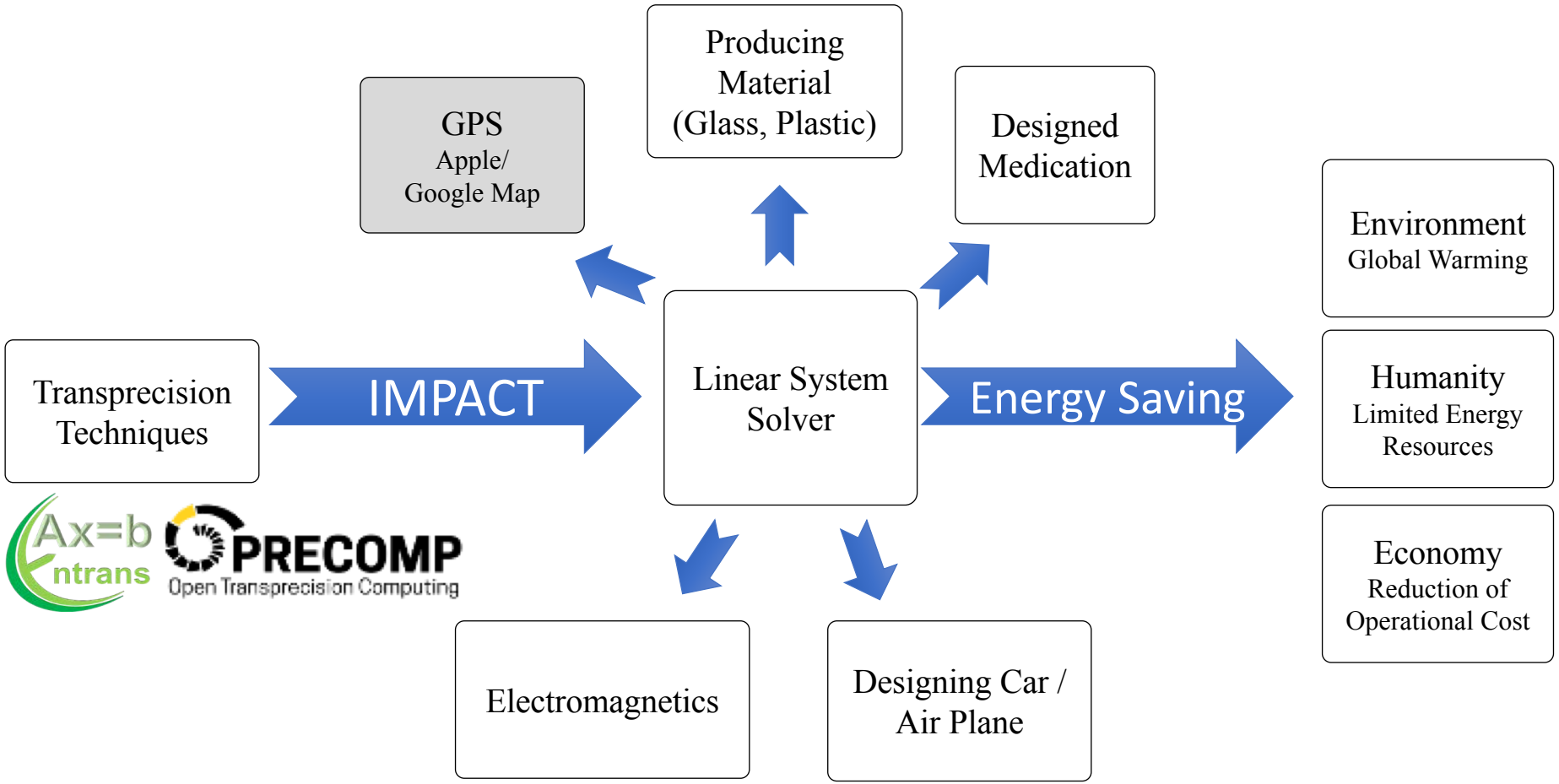


Size of Adder: Linear increase with precision

Size of Multiplier: Quadratic increase with precision

Part A. Linear Solver

Linear Solver Applications

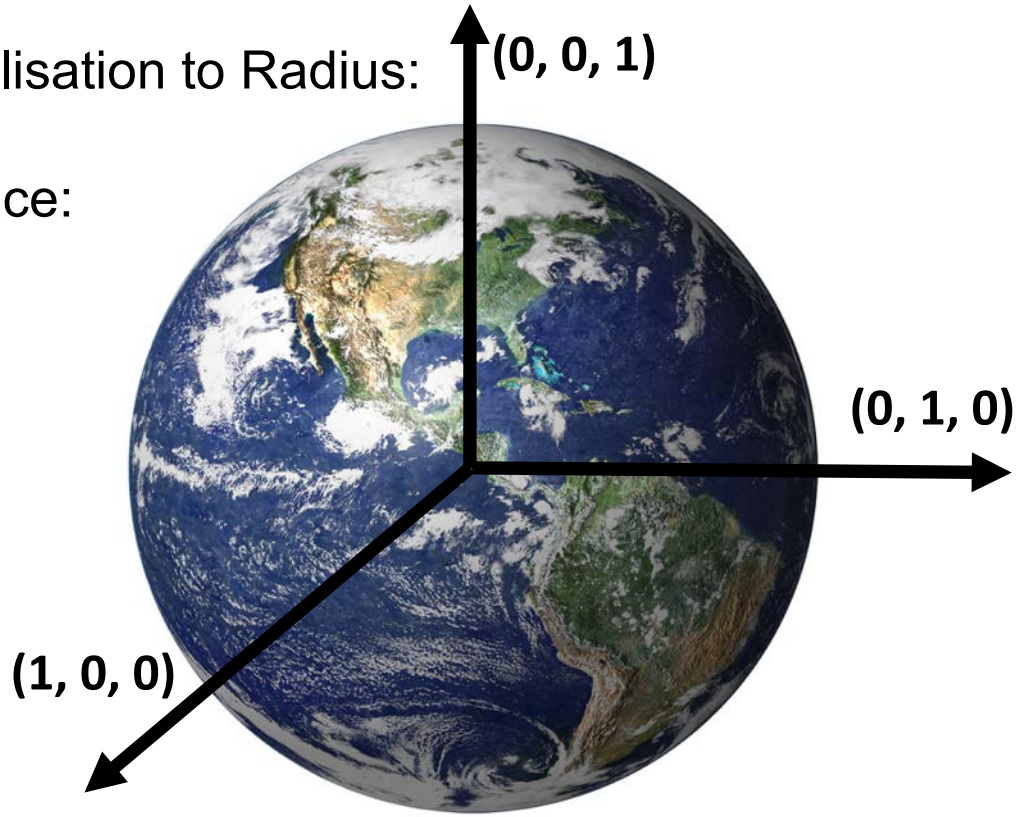


Global Positioning System (GPS)

Source (reference):
http://www.math.uri.edu/~jbaglama/classes/2011-2012/spring/mth215/project1.pdf

Positioning Normalisation to Radius:

Any point on Surface:
$x^2 + y^2 + z^2 = 1$

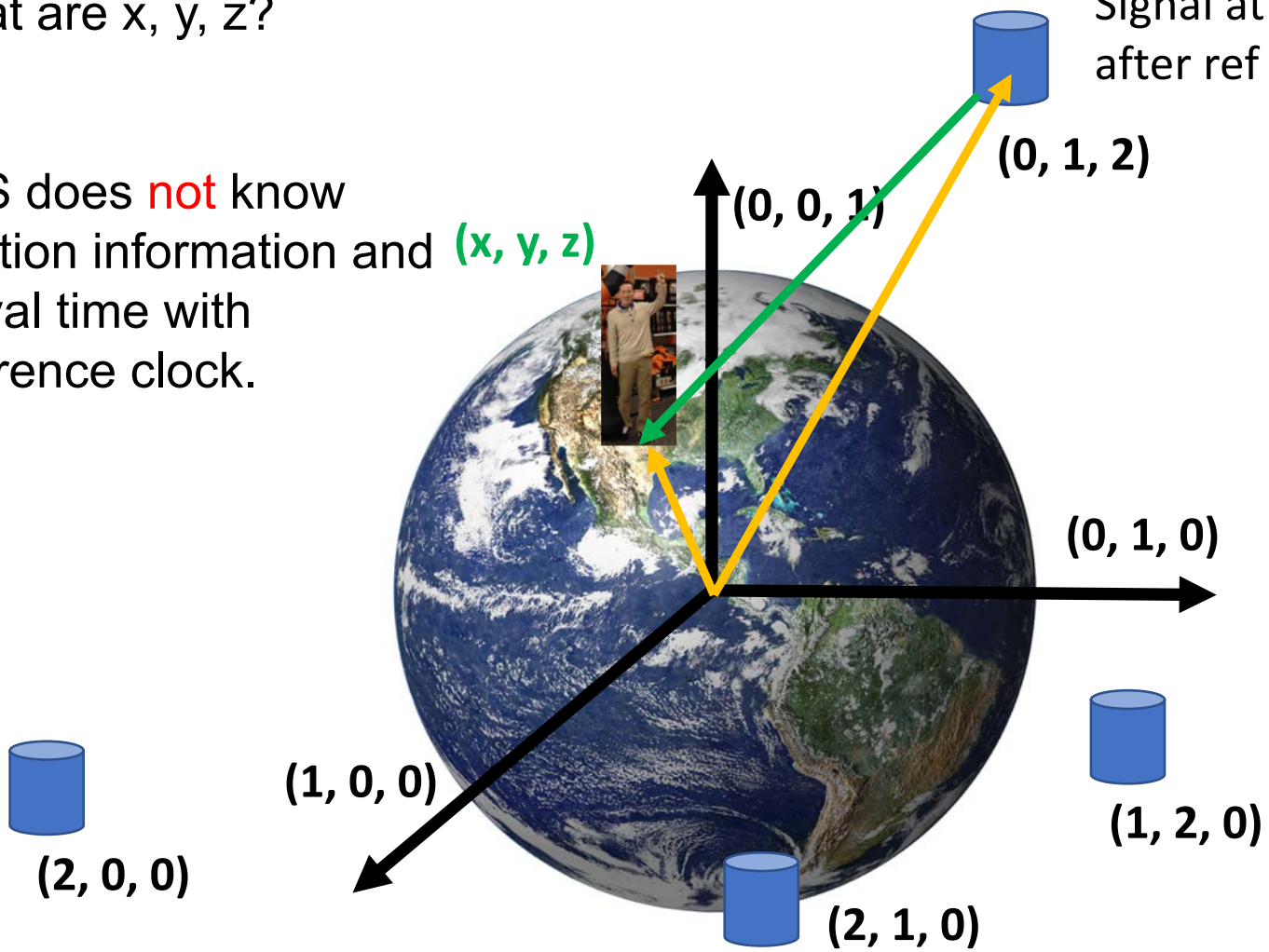


Global Positioning System (GPS)

What are x, y, z?

GPS does not know location information and arrival time with reference clock.

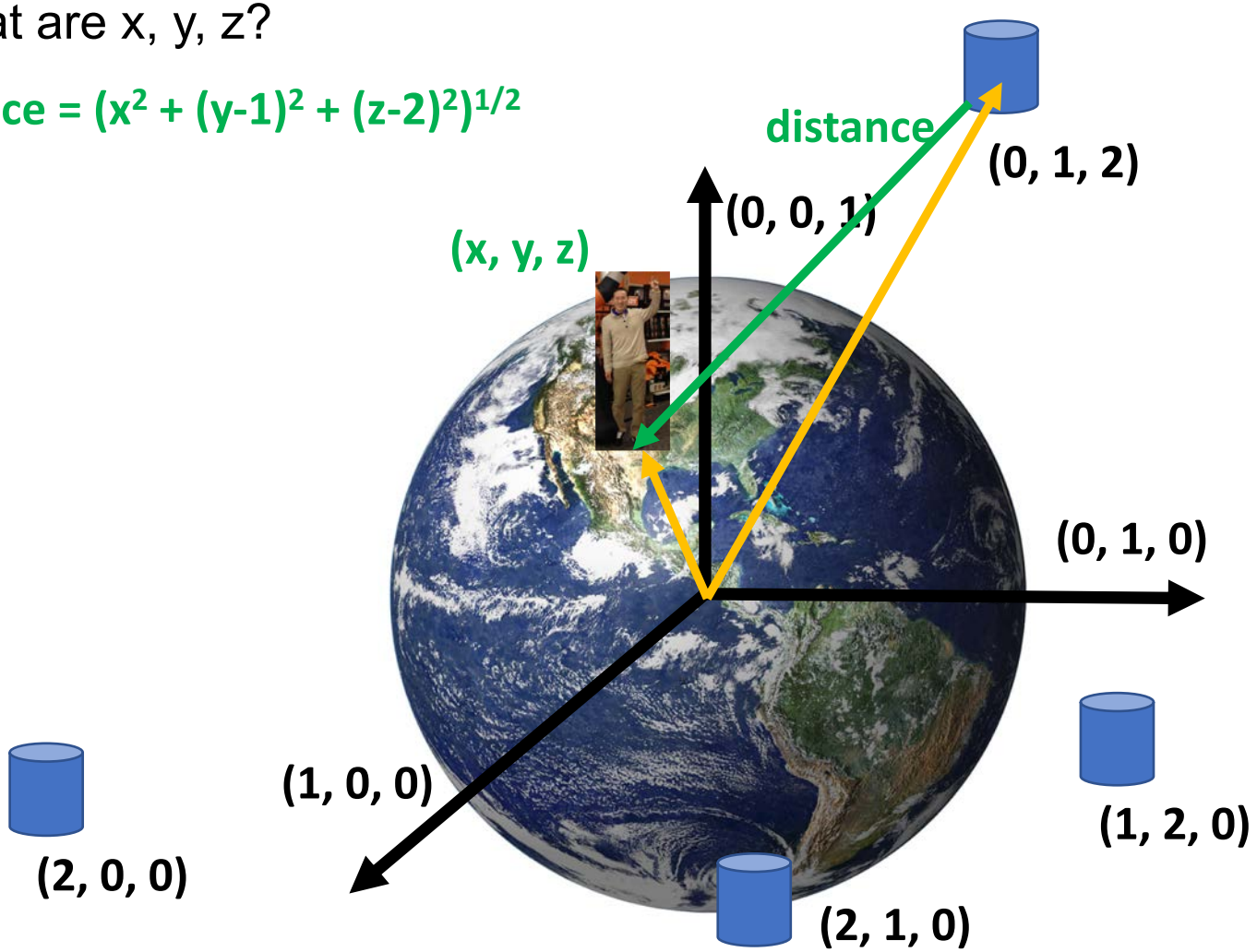
Signal at 2 seconds after ref clock 0.



Global Positioning System (GPS)

What are x, y, z?

distance = $(x^2 + (y-1)^2 + (z-2)^2)^{1/2}$



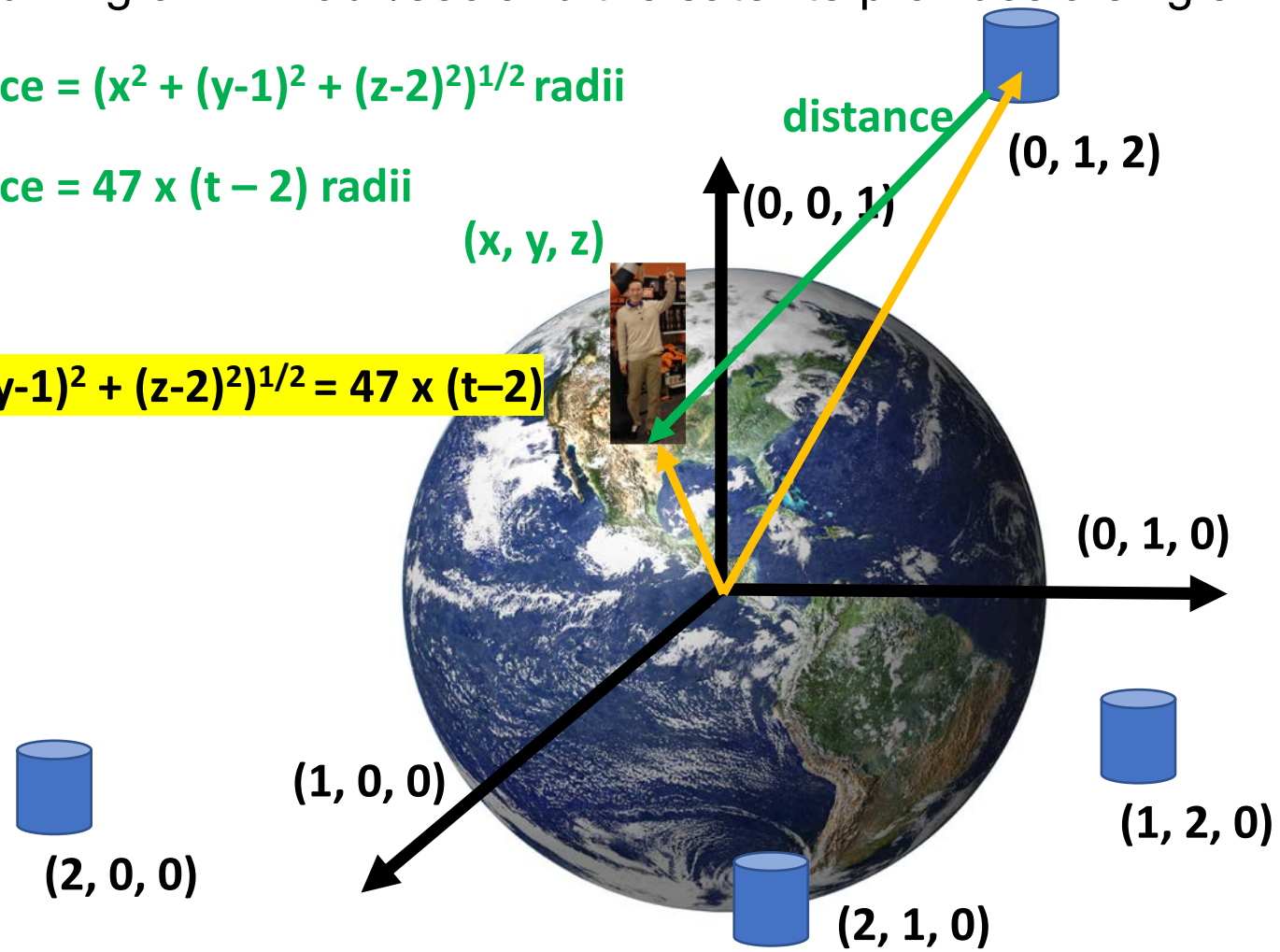
Global Positioning System (GPS)

Assuming c = 47 radii/sec and the satellite provides a single in t=2 seconds.

distance = $(x^2 + (y-1)^2 + (z-2)^2)^{1/2}$ radii

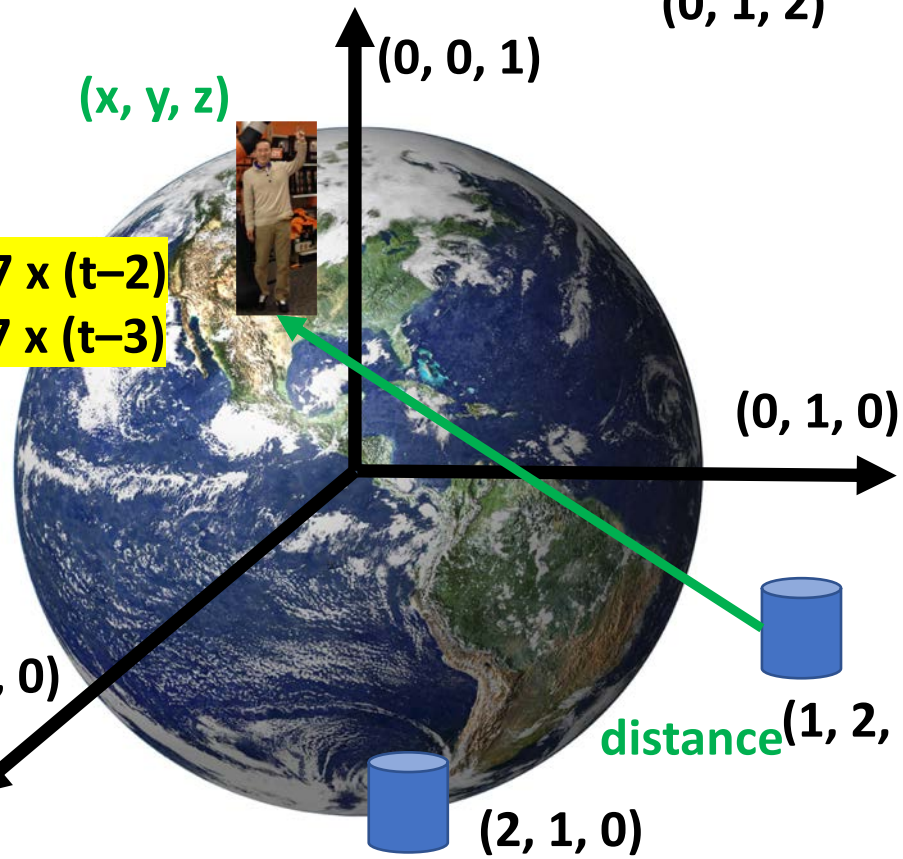
distance = 47 x (t – 2) radii

$(x^2 + (y-1)^2 + (z-2)^2)^{1/2} = 47 \times (t-2)$



Global Positioning System (GPS)

Assuming c = 47 radii/sec and the satellite provides a single in t=2 seconds.



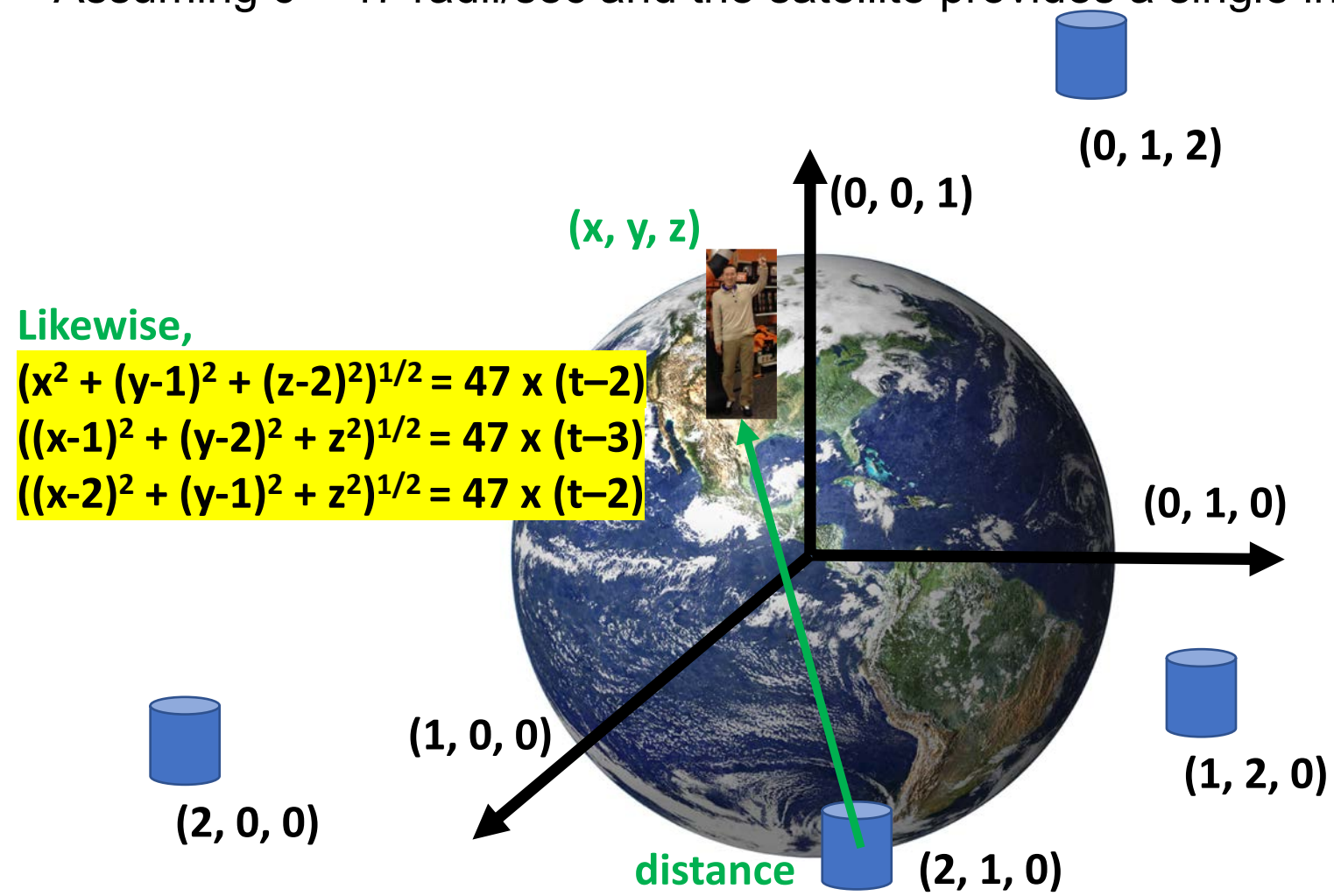
Likewise,

$(x^2 + (y-1)^2 + (z-2)^2)^{1/2} = 47 \times (t-2)$

$((x-1)^2 + (y-2)^2 + z^2)^{1/2} = 47 \times (t-3)$

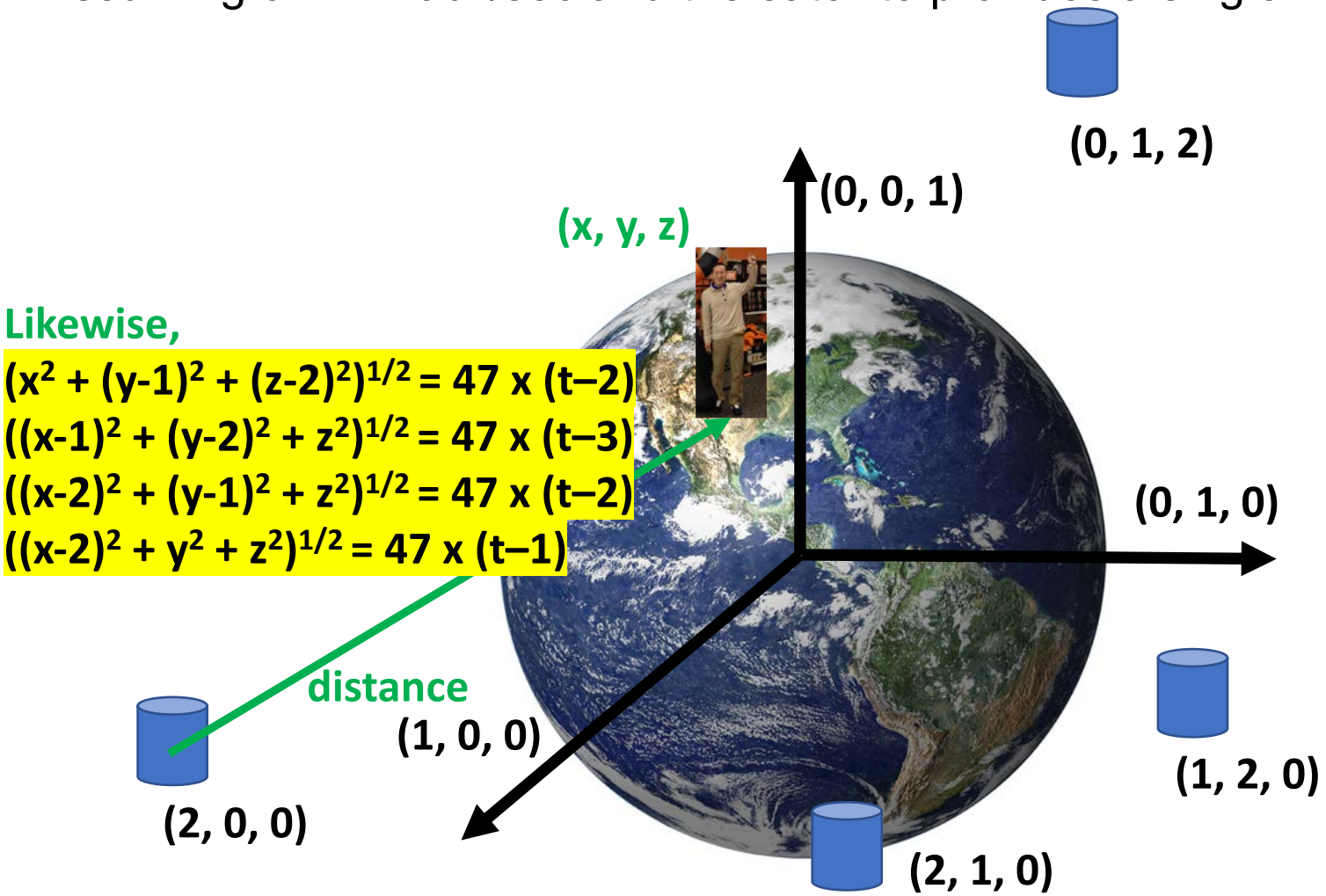
Global Positioning System (GPS)

Assuming c = 47 radii/sec and the satellite provides a single in t=2 seconds.



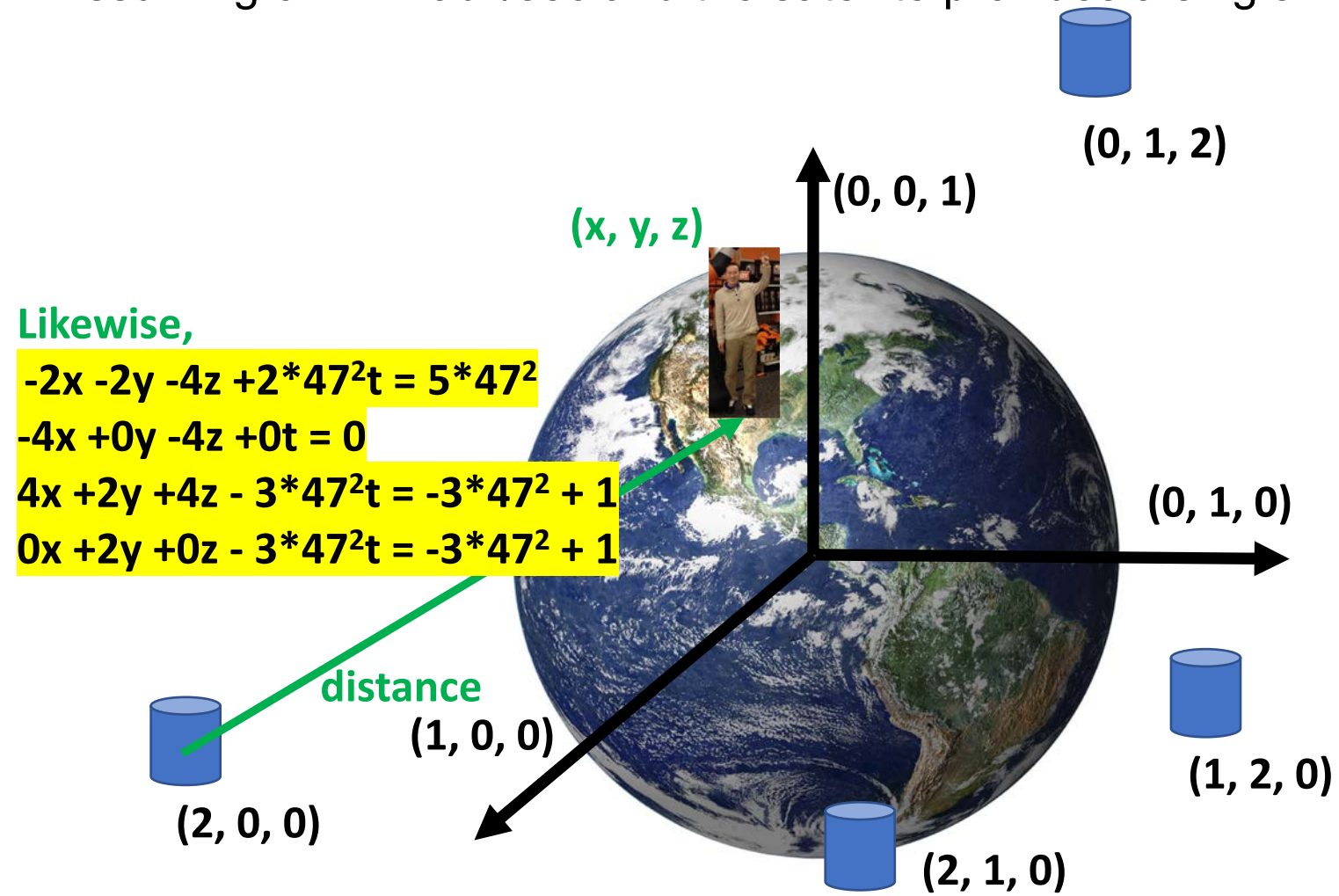
Global Positioning System (GPS)

Assuming c = 47 radii/sec and the satellite provides a single in t=2 seconds.



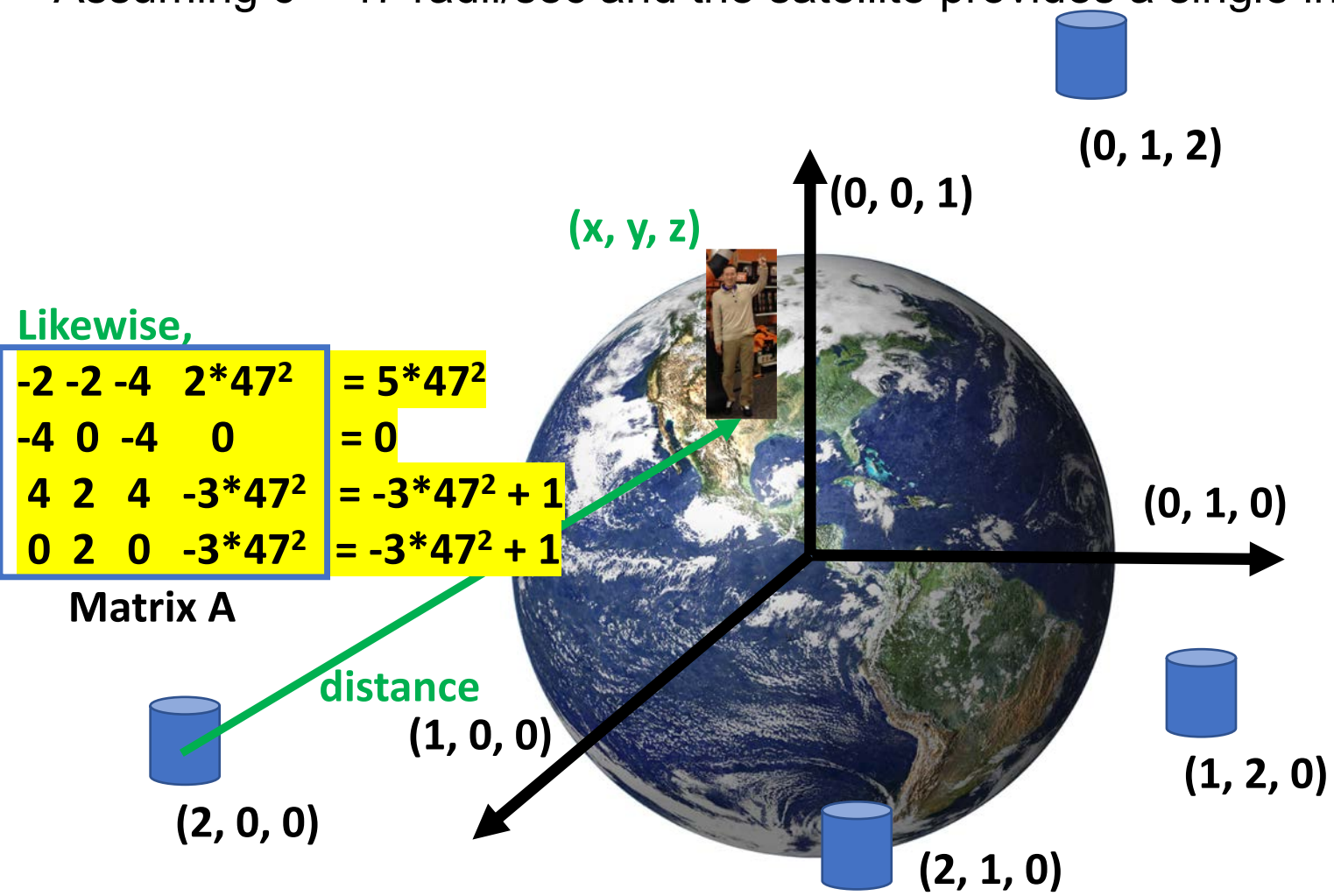
Global Positioning System (GPS)

Assuming c = 47 radii/sec and the satellite provides a single in t=2 seconds.



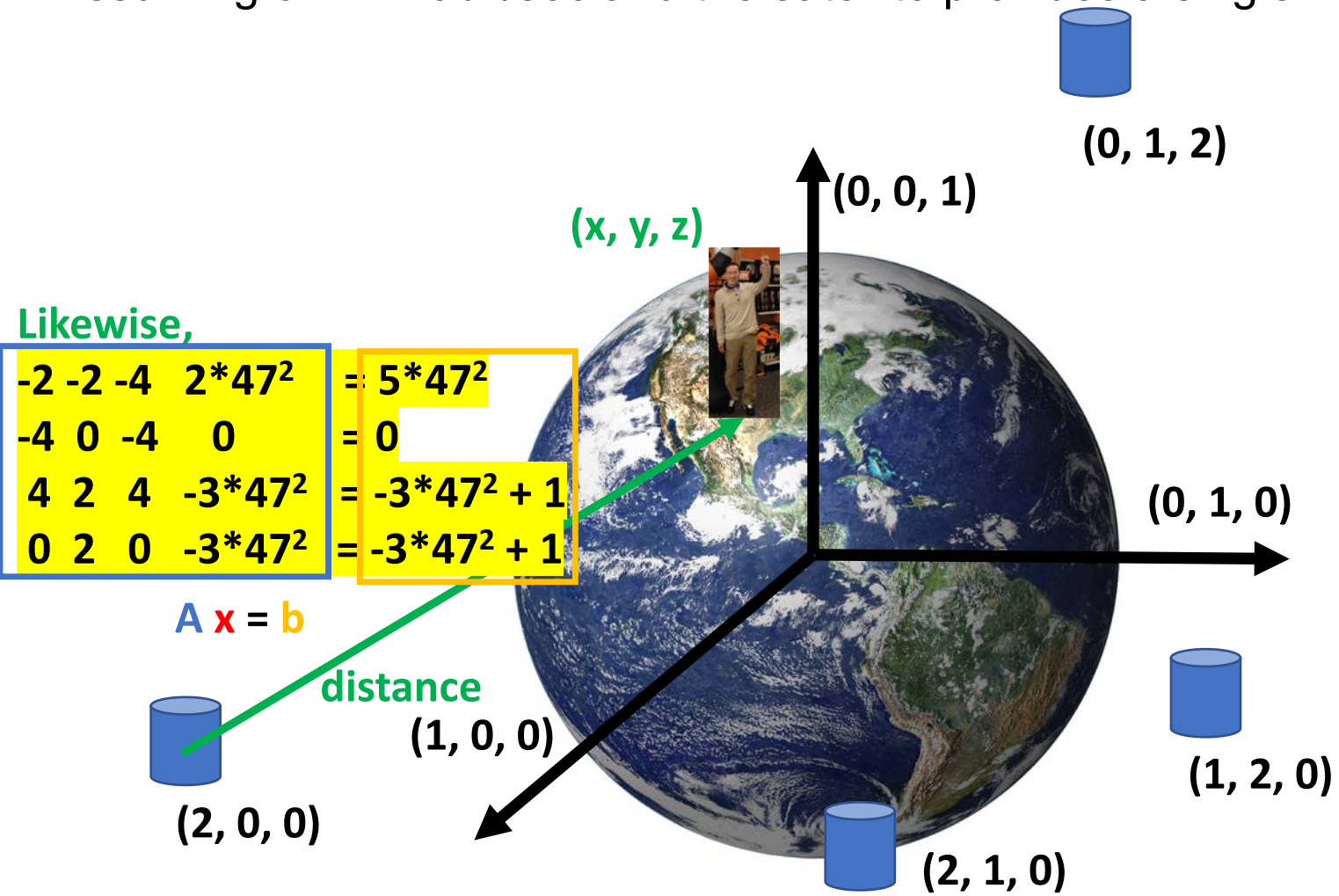
Global Positioning System (GPS)

Assuming c = 47 radii/sec and the satellite provides a single in t=2 seconds.



Global Positioning System (GPS)

Assuming c = 47 radii/sec and the satellite provides a single in t=2 seconds.



Linear Solver

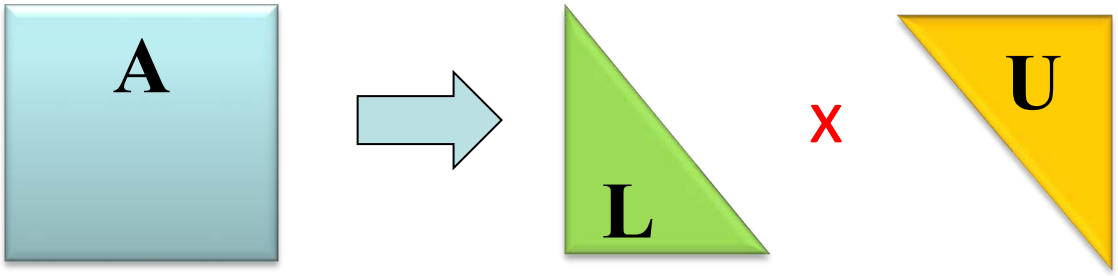
Linear Solver Algorithm

Find x in Ax = b!

Direct Solver (Ax = b)

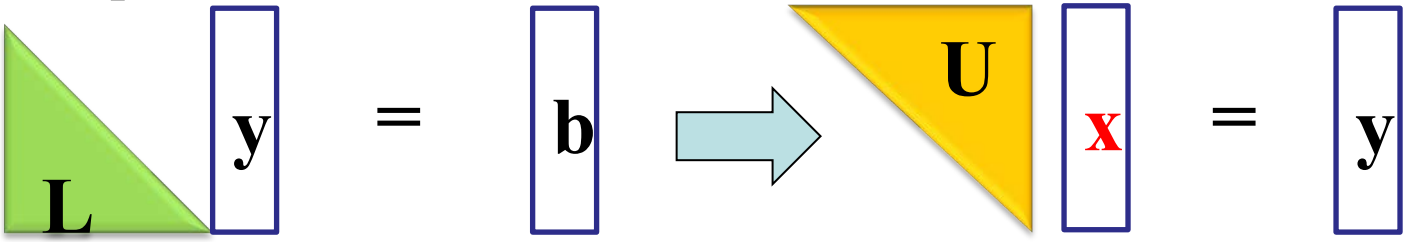
LU decomposition

Matrix Decomposition: 2/3 n^3 Ops



n^2 Ops (Forward Substitution)

n^2 Ops (Back Substitution)



LU Factorisation for 3 x 3 Matrix (i.e., Gaussian Elimination)

Gaussian Elimination eliminates a_{21}, a_{31}, a_{32}

$$\begin{bmatrix} 2 & 3 & 1 \\ 5 & 1 & 3 \\ 0 & 2 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \quad \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

At 2nd Row, multiply 5/2 to 1st Row and Subtract ; Store l_{21} <= 5/2

$$\begin{bmatrix} 2 & 3 & 1 \\ 5 & 1 & 3 \\ 0 & 2 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 3 & 1 \\ 0 & -13/2 & 1/2 \\ 0 & 2 & 4 \end{bmatrix}$$

LU Factorisation for 3 x 3 Matrix (i.e., Gaussian Elimination)

Gaussian Elimination eliminates a_{21}, a_{31}, a_{32}

$$\begin{bmatrix} 2 & 3 & 1 \\ 5 & 1 & 3 \\ 0 & 2 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 5/2 & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \quad \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

At 3rd Row, multiply 0 to 1st Row and Subtract ; Store l_{31} <= 0

$$\begin{bmatrix} 2 & 3 & 1 \\ 0 & -13/2 & 1/2 \\ 0 & 2 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 3 & 1 \\ 0 & -13/2 & 1/2 \\ 0 & 2 & 4 \end{bmatrix}$$

LU Factorisation for 3 x 3 Matrix (i.e., Gaussian Elimination)

Gaussian Elimination eliminates a_{21}, a_{31}, a_{32}

$$\begin{bmatrix} 2 & 3 & 1 \\ 5 & 1 & 3 \\ 0 & 2 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 5/2 & 1 & 0 \\ 0 & l_{32} & 1 \end{bmatrix} \quad \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

At 3rd Row, multiply -4/13 to 2st Row and Subtract ; Store l_{32} <= -4/13

$$\begin{bmatrix} 2 & 3 & 1 \\ 0 & -13/2 & 1/2 \\ 0 & 2 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 3 & 1 \\ 0 & -13/2 & 1/2 \\ 0 & 0 & 54/13 \end{bmatrix}$$

LU Factorisation for 3 x 3 Matrix (i.e., Gaussian Elimination)

Gaussian Elimination eliminates a_{21}, a_{31}, a_{32}

$$\begin{bmatrix} 2 & 3 & 1 \\ 5 & 1 & 3 \\ 0 & 2 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 5/2 & 1 & 0 \\ 0 & -4/13 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 1 \\ 0 & -13/2 & 1/2 \\ 0 & 0 & 54/13 \end{bmatrix}$$

At 3rd Row, multiply -4/13 to 2st Row and Subtract ; Store l_{32} <= -4/13

$$\begin{bmatrix} 2 & 3 & 1 \\ 0 & -13/2 & 1/2 \\ 0 & 2 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 3 & 1 \\ 0 & -13/2 & 1/2 \\ 0 & 0 & 54/13 \end{bmatrix}$$

Direct LU Solver (Ax = b)

Solve :

$$\begin{bmatrix} 2 & 3 & 1 \\ 5 & 1 & 3 \\ 0 & 2 & 4 \end{bmatrix} \mathbf{x} = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix}$$

Direct LU Solver (Ax = b)

Solve :

$$\begin{bmatrix} 2 & 3 & 1 \\ 5 & 1 & 3 \\ 0 & 2 & 4 \end{bmatrix} \mathbf{x} = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix} \Leftrightarrow$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 5/2 & 1 & 0 \\ 0 & -4/13 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 1 \\ 0 & -13/2 & 1/2 \\ 0 & 0 & 54/13 \end{bmatrix} \mathbf{x} = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix}$$

Direct LU Solver (Ax = b)

Solve : seek y_1, y_2, y_3 by forward substitutions

$$\begin{bmatrix} 2 & 3 & 1 \\ 5 & 1 & 3 \\ 0 & 2 & 4 \end{bmatrix} \mathbf{x} = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 5/2 & 1 & 0 \\ 0 & -4/13 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 1 \\ 0 & -13/2 & 1/2 \\ 0 & 0 & 54/13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix} \quad \Leftrightarrow$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 5/2 & 1 & 0 \\ 0 & -4/13 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix}$$

Direct LU Solver (Ax = b)

Solve : seek x_1, x_2, x_3 by backward substitutions

$$\begin{bmatrix} 2 & 3 & 1 \\ 5 & 1 & 3 \\ 0 & 2 & 4 \end{bmatrix} \mathbf{x} = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 5/2 & 1 & 0 \\ 0 & -4/13 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 1 \\ 0 & -13/2 & 1/2 \\ 0 & 0 & 54/13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix} \Leftrightarrow$$

$$\begin{bmatrix} 2 & 3 & 1 \\ 0 & -13/2 & 1/2 \\ 0 & 0 & 54/13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 6 \\ 24/13 \end{bmatrix}$$

Direct LU Solver Accuracy

Accuracy: $||x - x^*||/ ||x^*|| = \phi(n)\, \kappa(A)\, \epsilon_0$

What is condition number $\kappa(A)$**?**
Sensitivity of Solution to perturbation: $1 \leq \kappa(A) < \infty$

ϵ_0 **=>** 2^{-24} **single precision,** 2^{-53} **double precision**

Accuracy of LU solver is not enough

Iterative Refinement with LU Solver

Accuracy: $\|x - x^*\| / \|x^*\| = \phi(n)\, \kappa(A)\, \epsilon_0$

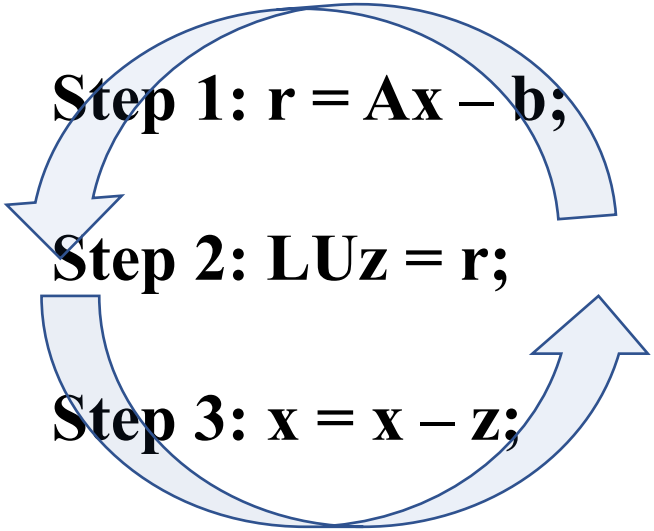
What is condition number $\kappa(A)$?
Sensitivity of the matrix A to perturbation: $1 \leq \kappa(A) < \infty$

ϵ_0 => 2^{-24} single precision, 2^{-53} double precision

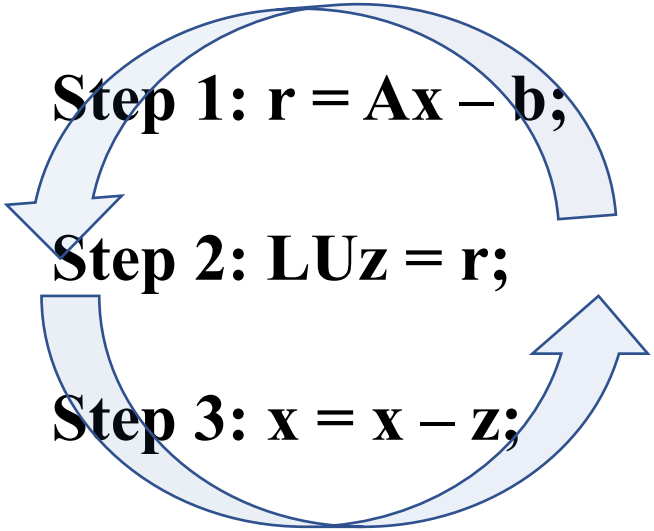
Accuracy of LU solver is not enough
=> Use it as an approximator inside Iterative Refinement

Iterative Refinement(IR): $q = \|x - x^*\| / \|x^*\|$, $q < 1$ **for IR Success**

Iterative Refinement with LU Solver



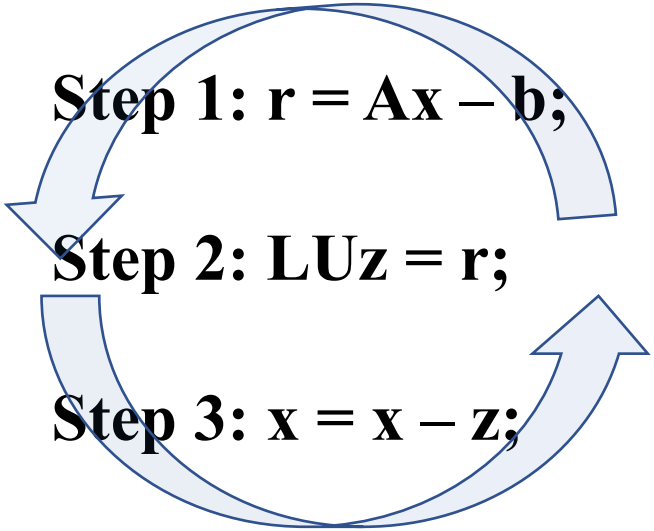
Iterative Refinement with LU Solver



Step 1:

Seeks **r**: r = A(*Δ*x); Matrix times previous solution error sought

Iterative Refinement with LU Solver



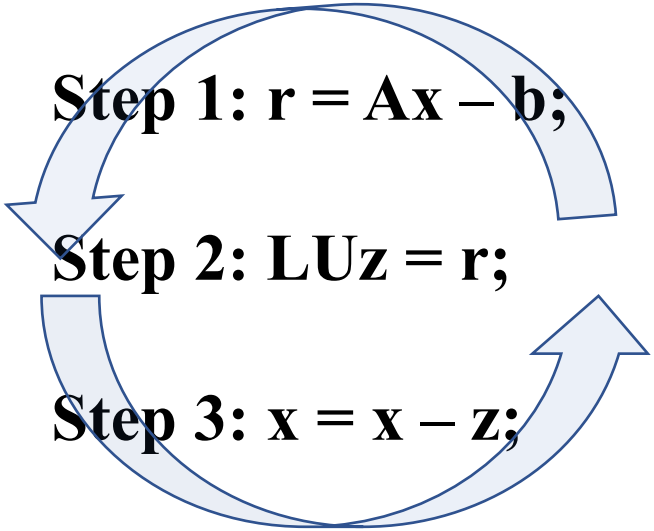
Step 1:

Seeks **r**: r = A(Δx); Matrix times previous solution error sought

Step 2:

Seeks **z**: z = A^{-1}A(Δx); previous solution error approximation sought

Iterative Refinement with LU Solver



Step 1:

Seeks **r**: r = A(Δx); Matrix times previous solution error sought

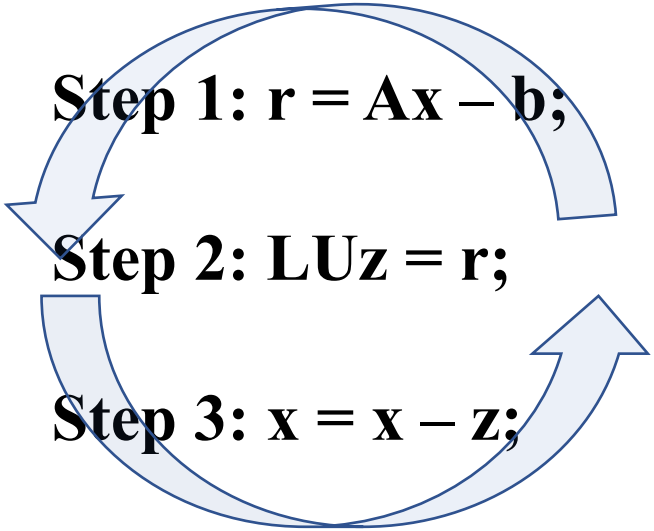
Step 2:

Seeks **z**: z = A^{-1}A(Δx); previous solution error approximation sought

Step 3:

Update **x**: x=x-(Δx);previous solution error approximation subtracted

Conventional Iterative Refinement



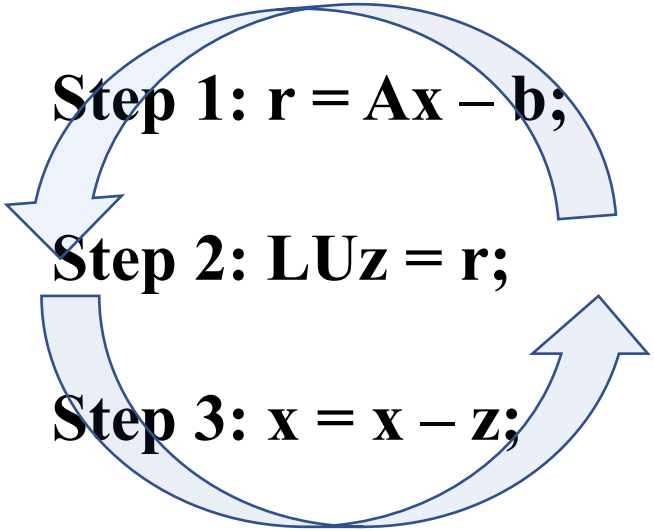
Step 1: r = Ax – b; **double-double; 2^{-106}**

Step 2: LUz = r; **double; 2^{-53}**

Step 3: x = x – z; **double-double; 2^{-106}**

Accuracy: $\|x - x^*\| / \|x^*\| < 2^{-53}$

Conventional Iterative Refinement



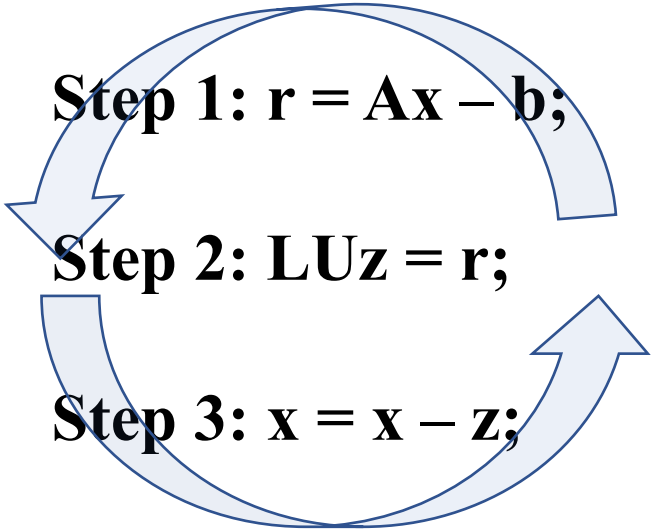
Step 1: r = Ax – b; **double-double; 2^{-106}**

Step 2: LUz = r; **double; 2^{-53}**

Step 3: x = x – z; **double-double; 2^{-106}**

Accuracy: $\|x - x^*\| / \|x^*\| < 2^{-53} \sim 10^{-16}$

Conventional Iterative Refinement



Step 1: r = Ax – b; **double-double; 2^{-106}**

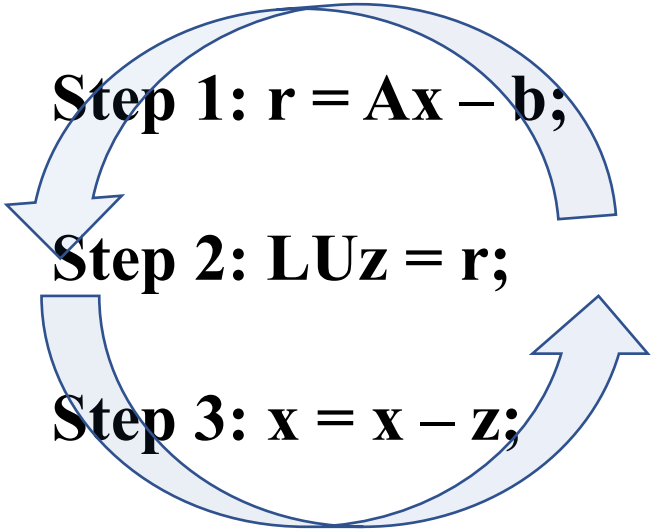
Step 2: LUz = r; **double; 2^{-53}**

Step 3: x = x – z; **double-double; 2^{-106}**

Accuracy: $||x - x^*||/ ||x^*|| < 2^{-53}$

Can we apply reduced precision arithmetic without losing accuracy? -> Transprecision Technique

Conventional Iterative Refinement



Step 1: r = Ax – b; **double-double; 2^{-106}**

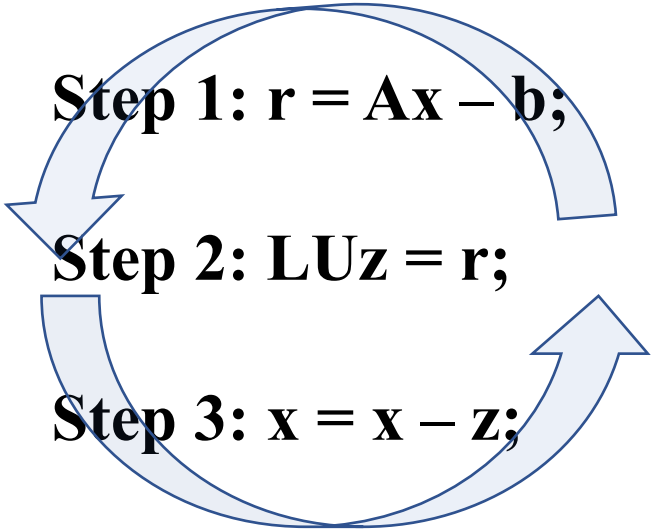
Step 2: LUz = r; **double; 2^{-53}**

Step 3: x = x – z; **double-double; 2^{-106}**

Accuracy: $||x - x^*||/ ||x^*|| < 2^{-53}$

TT1: $||z - z^*||/ ||z^*|| < 1$ for LU solver in Step 2.

Mixed Precision Iterative Refinement



Step 1: r = Ax – b; double-double; 2^{-106}

Step 2: LUz = r; single; 2^{-24}

Step 3: x = x – z; double-double; 2^{-106}

Accuracy: $||x - x^*||/ ||x^*|| < 2^{-53}$

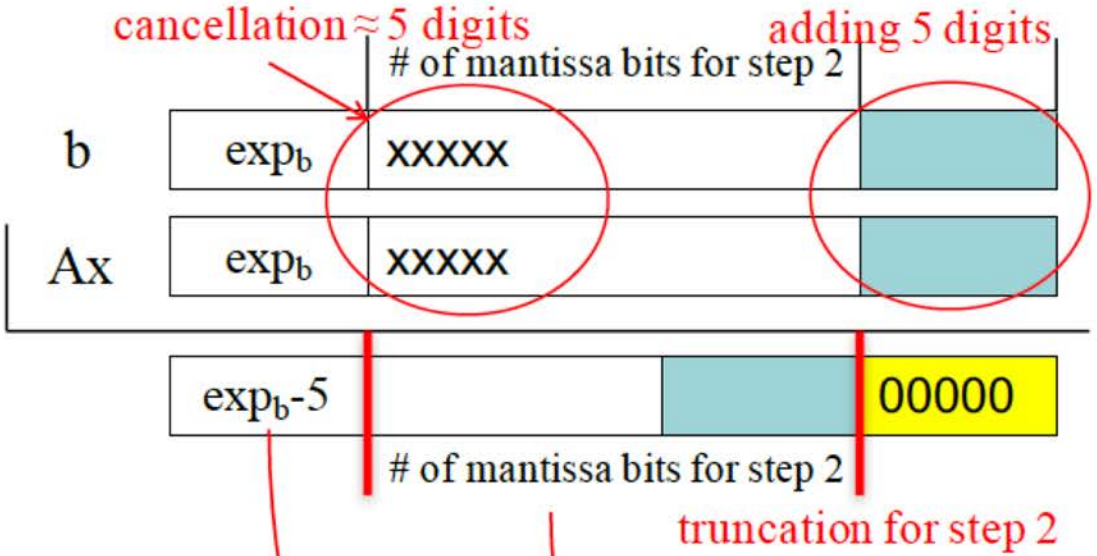
TT1: $||z - z^*||/ ||z^*|| < 1$ for LU solver in Step 2.

=> This is generally referred to Mixed Precision IR.

Transprecision Iterative Refinement

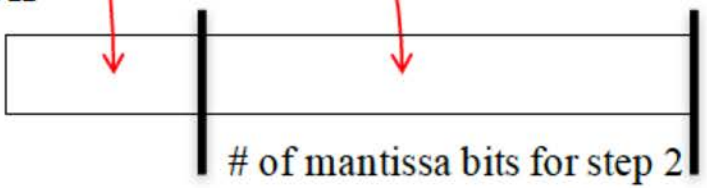
seeking residual

step 1 $r^{(i)} = A\,x^{(i)} - b$

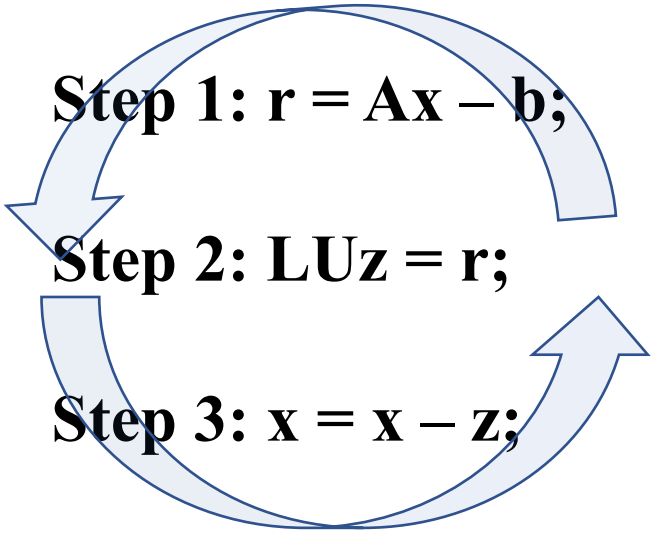


seeking the error at the previous iteration

step 2 $A\,z^{(i)} = r^{(i)}$



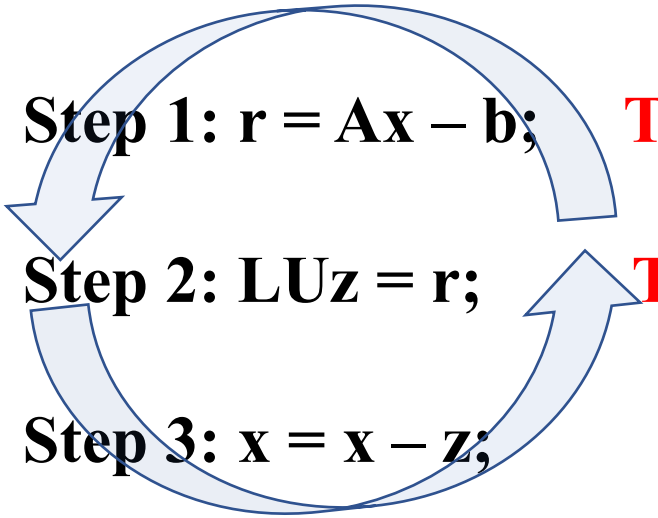
Transprecision Iterative Refinement



Step 1: r = Ax – b; double-double; 2^{-106}

Step 2: LUz = r; double; 2^{-53}

Step 3: x = x – z; double-double; 2^{-106}



Step 1: r = Ax – b; TT2:double=>double-double;2^{-53}=>2^{-106}

Step 2: LUz = r; TT1:single; 2^{-24}

Step 3: x = x – z; double-double; 2^{-106}

Transprecision Iterative Refinement

Beyond this lecture, other two transprecision techniques are employed.

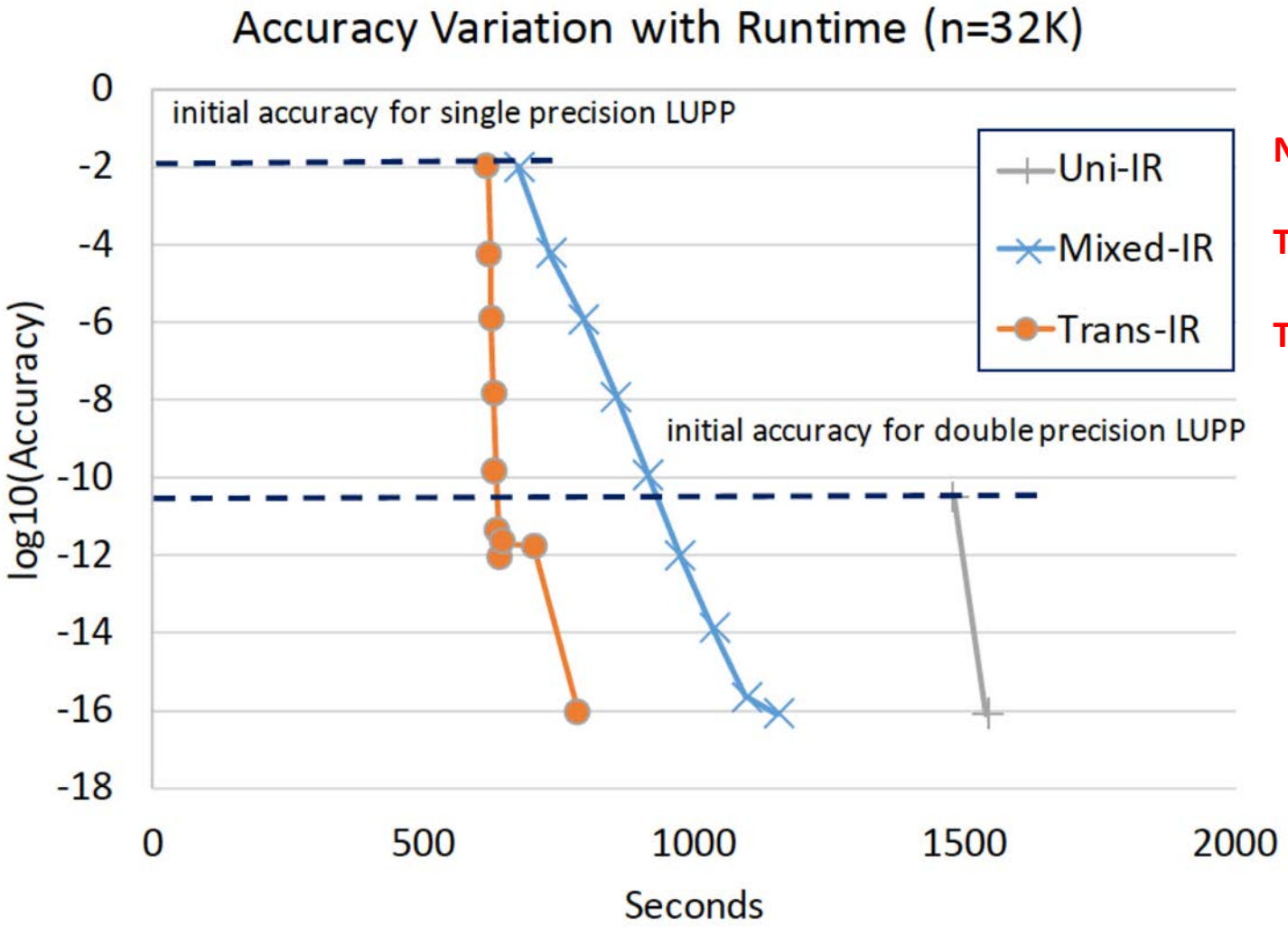
TT3: Recursive iterative refinement

TT4: Skipping accuracy checks

Reference for details of all TTs:

J. Lee, H. Vandierendonck, M. Arif, G. D. Peterson and D. S. Nikolopoulos, "Energy-Efficient Iterative Refinement using Dynamic Precision," in *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*. doi: 10.1109/JETCAS.2018.2850665

Impact of Transprecision Iterative Refinement

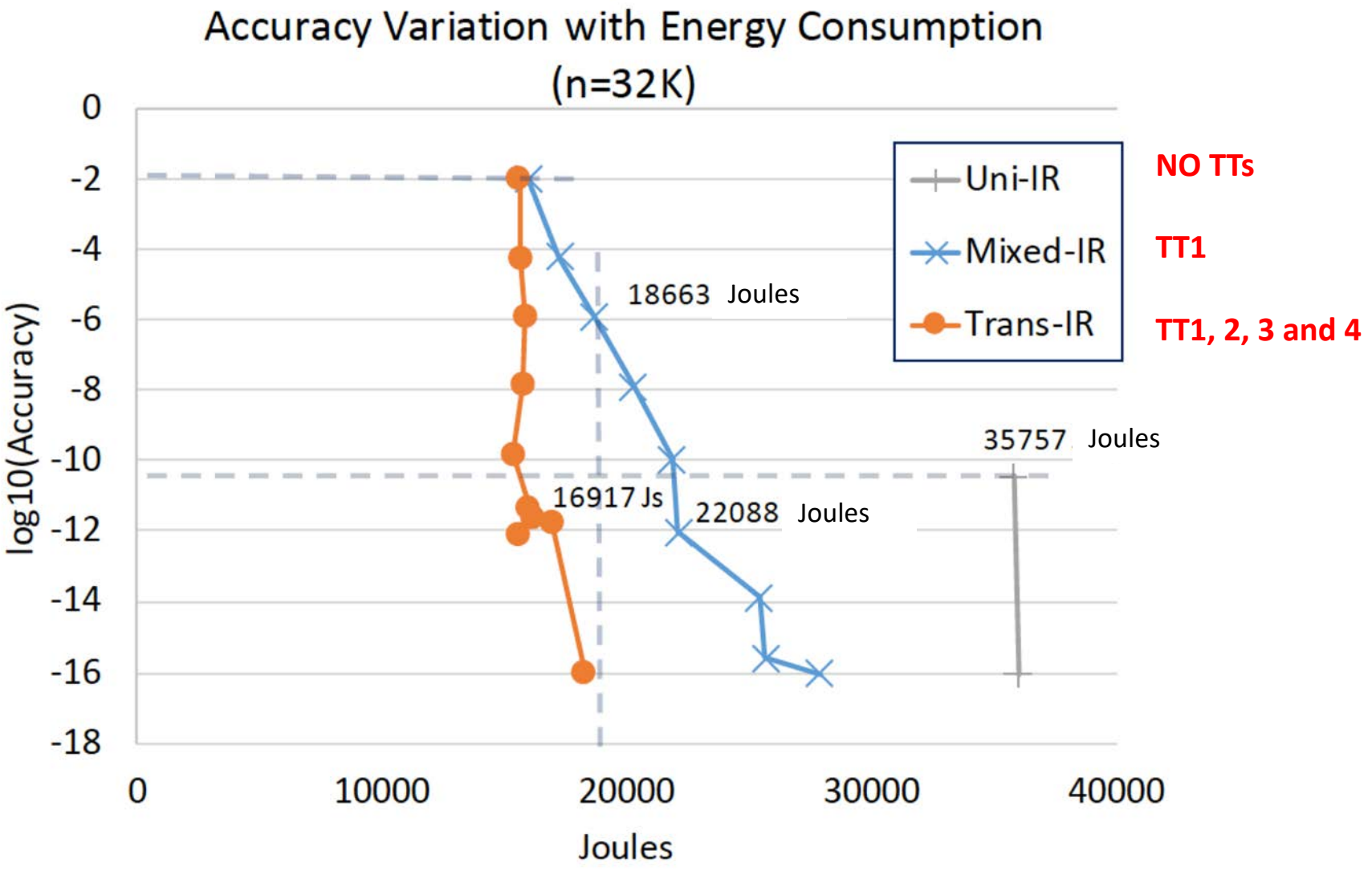


NO TTs

TT1

TT1, 2, 3 and 4

Impact of Transprecision Iterative Refinement

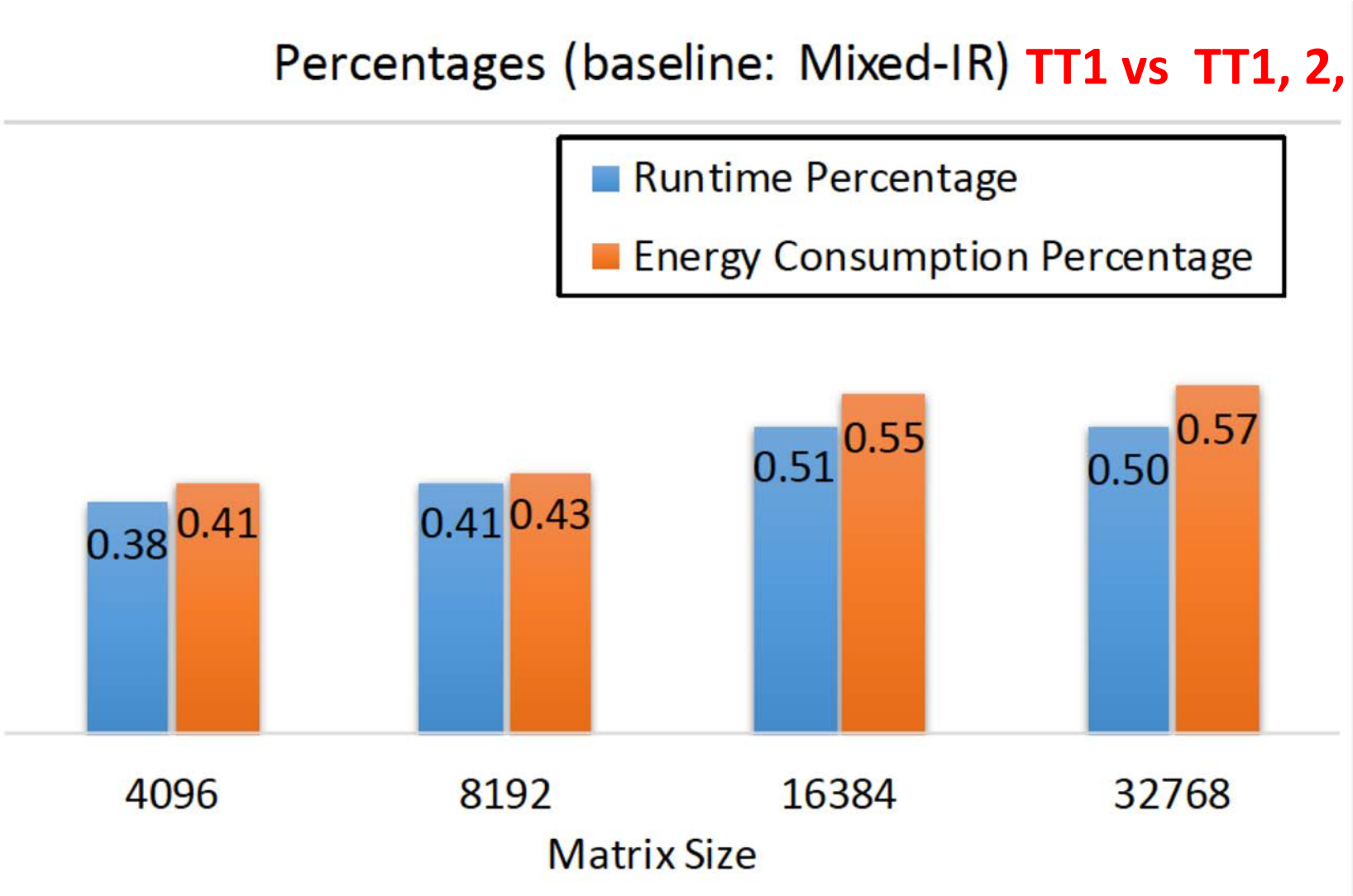


NO TTs

TT1

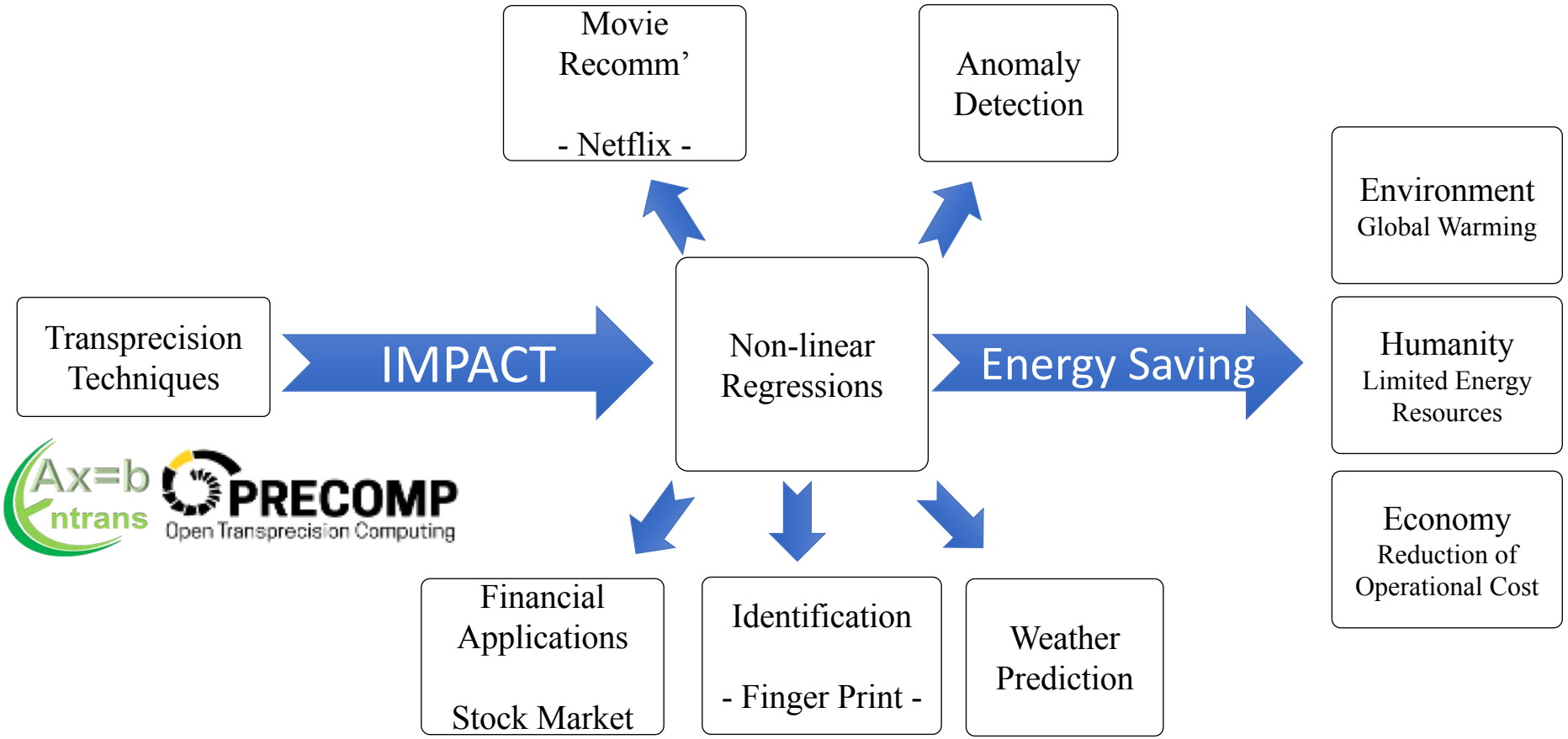
TT1, 2, 3 and 4

Impact of Transprecision Iterative Refinement



Part B. Non-linear Regressions

Impact of Transprecision on Non-linear Regression Applications



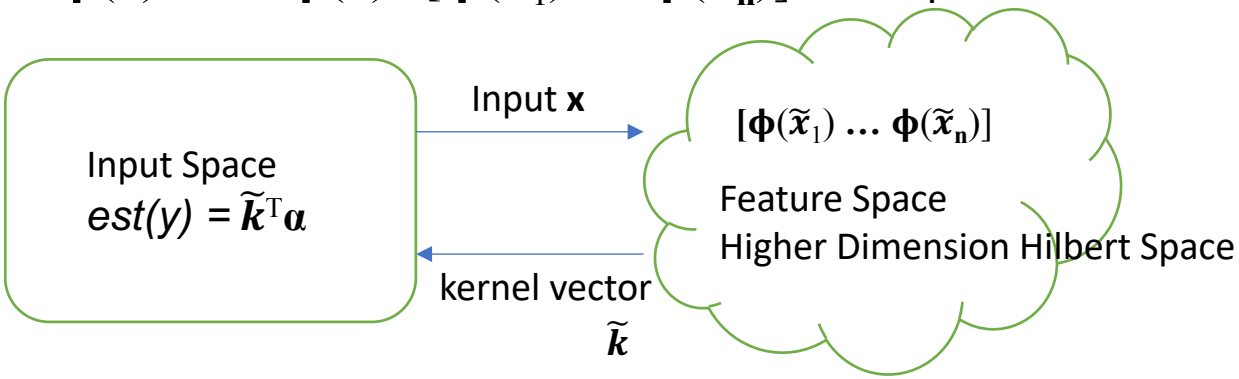
Linear Regression and Non-linear Regression

Linear Regression (e.g., Recursive Least Squares) :
Seek $\mathbf{w} = (w_0, w_1, w_2, w_3)$ to estimate y given an input $\mathbf{x} = (1, x_1, x_2, x_3)$ with

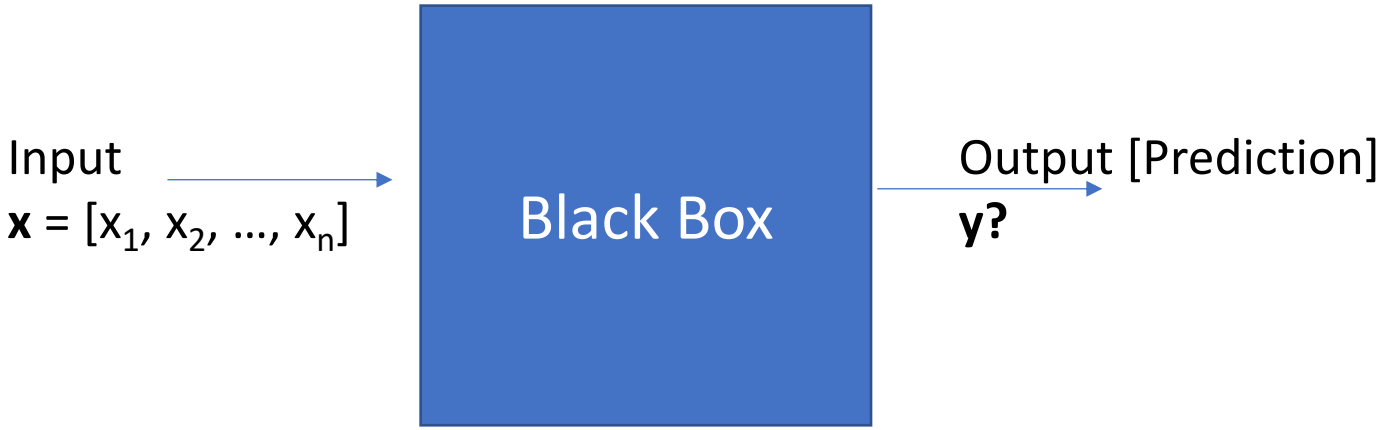
$$est(y) = \mathbf{x}^T\mathbf{w} = w_0\ 1 + w_1\ x_1 + w_2\ x_2 + w_3\ x_3.$$

Non-linear Regression using Kernel Method
(e.g., Kernel Recursive Least Squares):
Perform regression in higher dimension Hilbert space $\mathcal{H}$

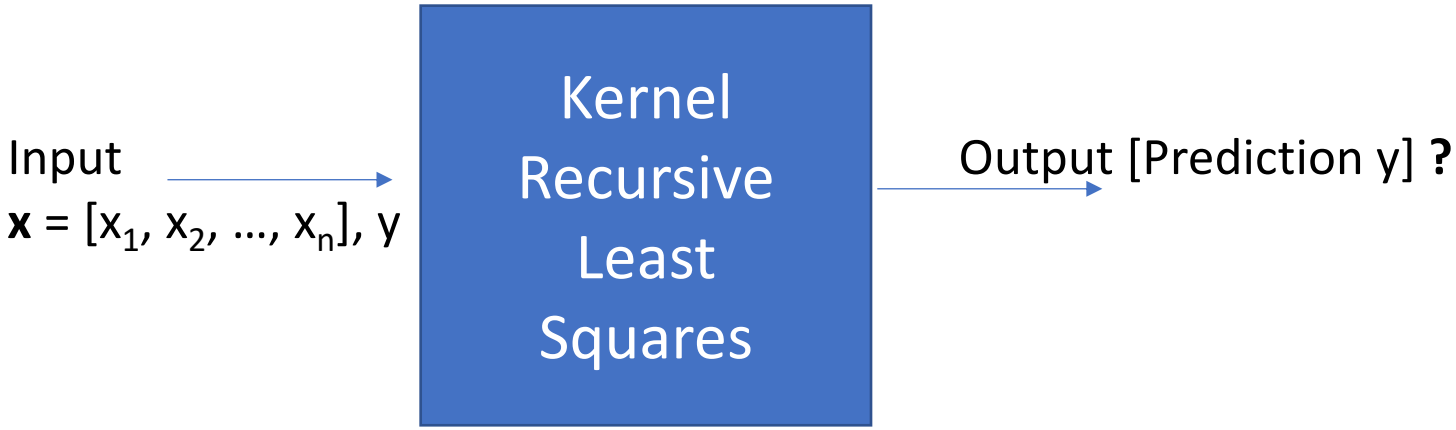
$$est(y) = \boldsymbol{\phi}(\mathbf{x})^T\ \mathbf{w} = \boldsymbol{\phi}(\mathbf{x})^T\ [\boldsymbol{\phi}(\tilde{\boldsymbol{x}}_1)\ \ldots\ \boldsymbol{\phi}(\tilde{\boldsymbol{x}}_n)]\ \boldsymbol{\alpha} = \tilde{\boldsymbol{k}}_.^T\ \boldsymbol{\alpha}$$



Non-linear Regression Machine Learning

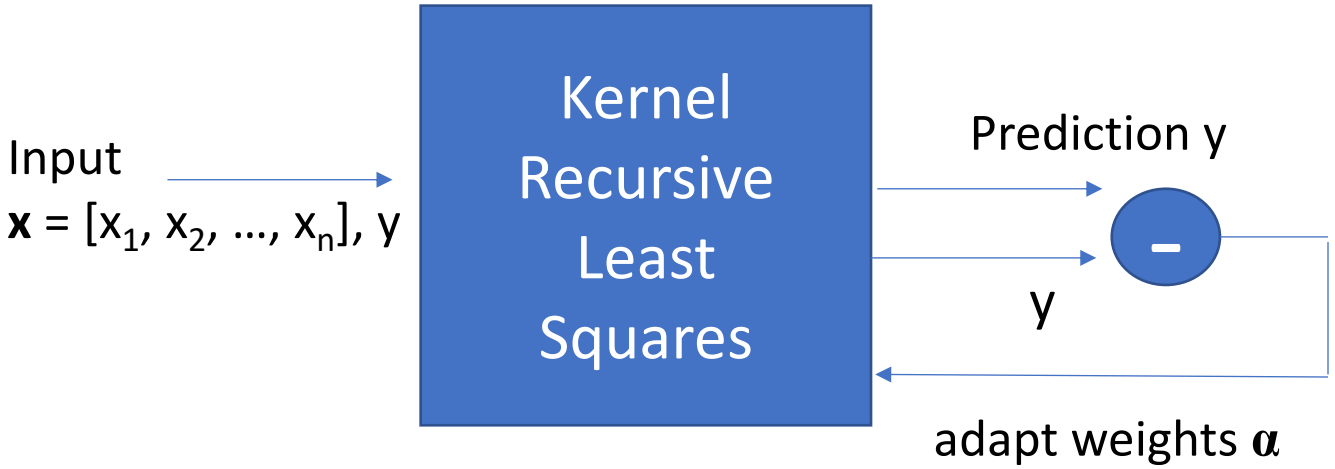


Non-linear Regression Machine Learning



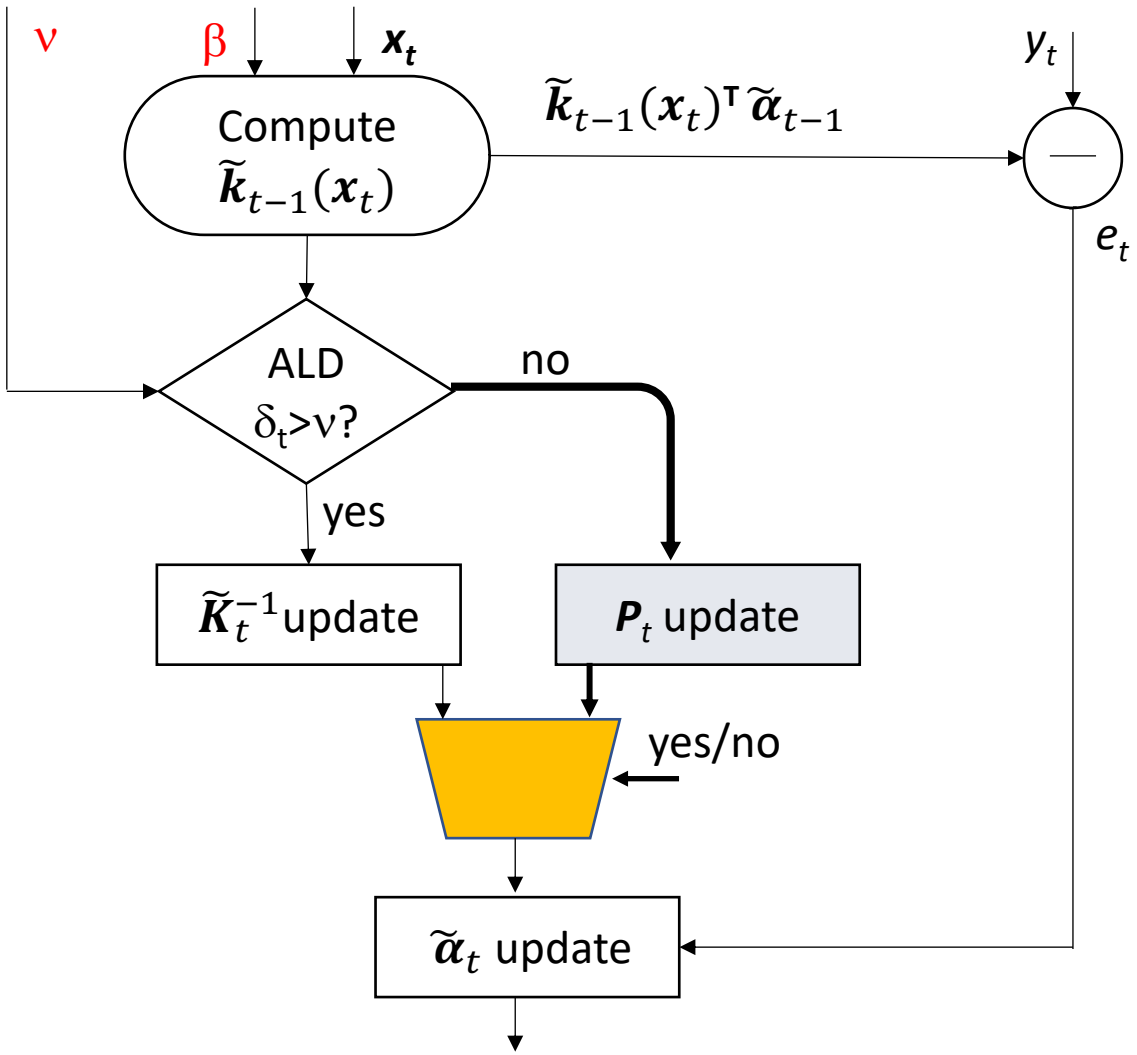
Realtime Training

Non-linear Regression Machine Learning



Realtime Training
(i.e., Kernel Adaptive Filter)

KRLS Algorithm Computing Components



Kernel Recursive Least Squares (KRLS)

Applications: Non-linear regressions

- Simple RLS mechanism

- No local minima issue

- Fast Convergence / Good Prediction Accuracy

- Online Adaptive Learning (Learning weights One by One – Fit for large scale Machine Learning)

Model Selection for KRLS

KRLS model depending on two hyperparameters :
Approx Linear Dependency(ALD) threshold ν, Kernel Width β

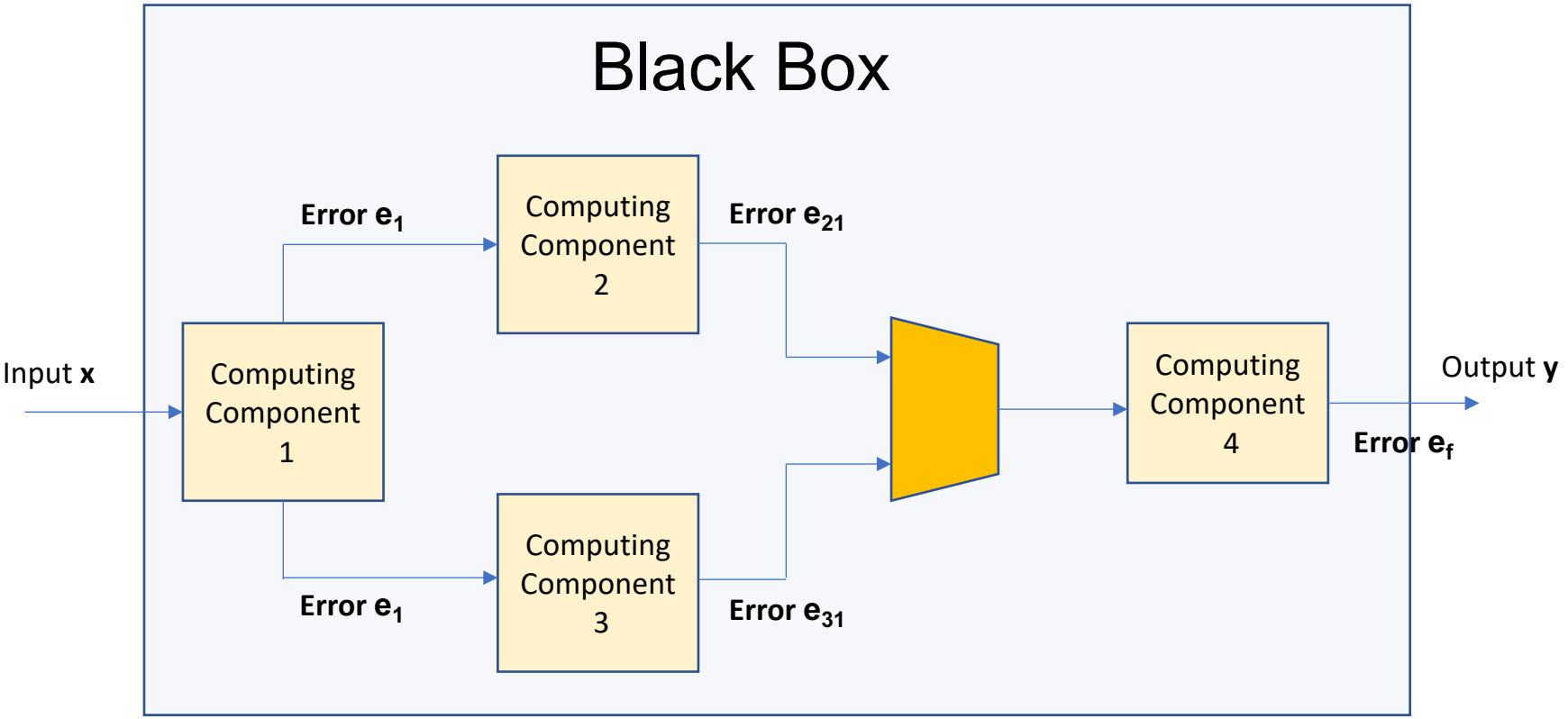
ALD threshold ν: Generalization and Approximation level

Kernel Width β: Inner product distribution

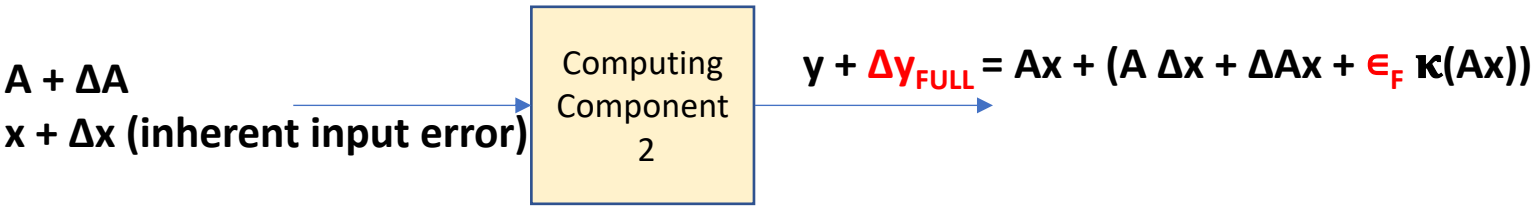
$$\kappa(\mathbf{x_t}, \mathbf{x_i}) =< \phi(\mathbf{x_t}), \phi(\mathbf{x_i}) >= exp^{-(\mathbf{x_t}-\mathbf{x_i})^T(\mathbf{x_t}-\mathbf{x_i})/(2\beta^2)}$$

Optimal ν, β depends on Inputs
In many cases, Cross Validation decides Optimal ν, β

Error propagation for each computing component

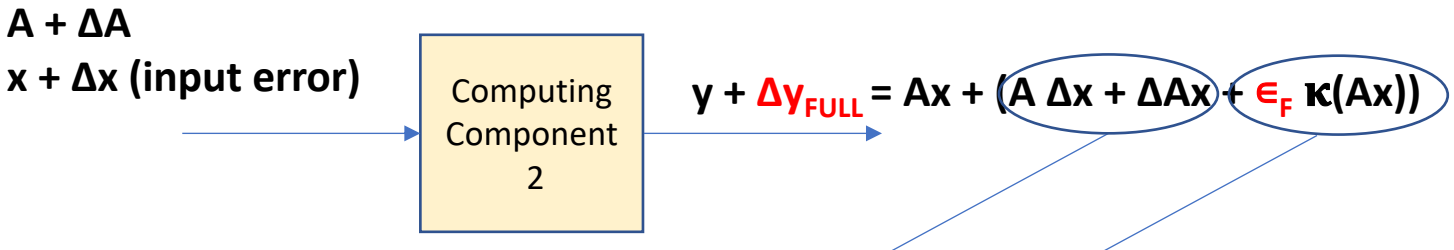


Error propagation for each computing component



Full Precision Arithmetic

Error propagation for each computing component

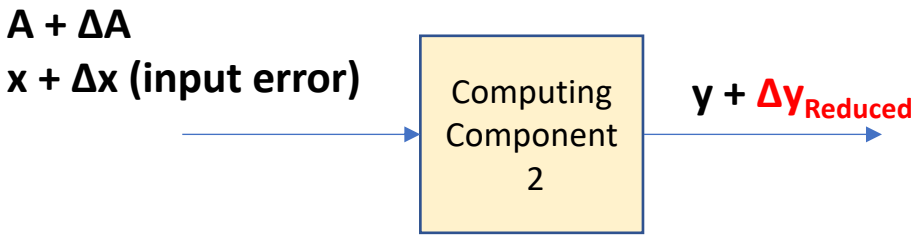


$||\Delta y_{FULL}||$ includes

err_{una}(Unavoidable) and err_{rnd} (Controllable: Precision arithmetic related)

Key Idea: ∈ can be balanced to err_{una}.
Employ lower precision arithmetic ∈ until $||\Delta y_{FULL}||$ not affected

Error propagation for each computing component



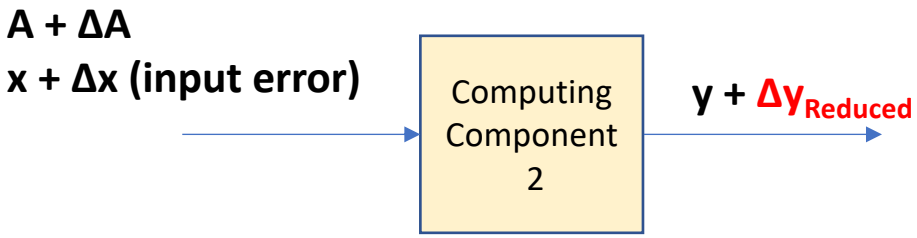
Reduced Precision Arithmetic

$$err_{una} \gtrsim err_{rnd}$$

When size of Unavoidable Error equivalent of size of rounding off error

=> Reduced Arithmetic can be applied to the Computing Component

Error propagation for each computing component



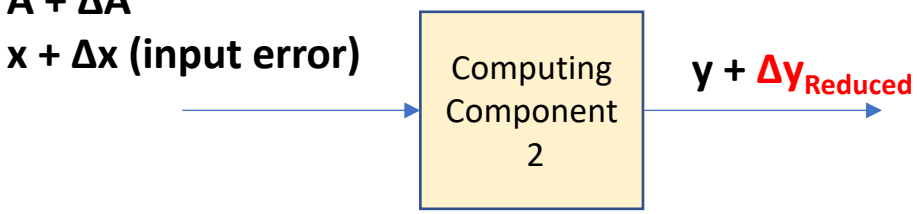
Reduced Precision Arithmetic

$$||A\Delta x + \Delta A x|| \gtrsim ||\epsilon_R \kappa(Ax)||$$

(unavoidable err) (controllable rounding-off error)

Reduced Arithmetic can be applied to the Computing Component

Error propagation for each computing component



Case 1: Linear Solver

$err_{rnd} \propto \kappa(A)\epsilon_R$

Case 2: Matrix-vector

$err_{rnd} \propto \kappa(A)\epsilon_R$

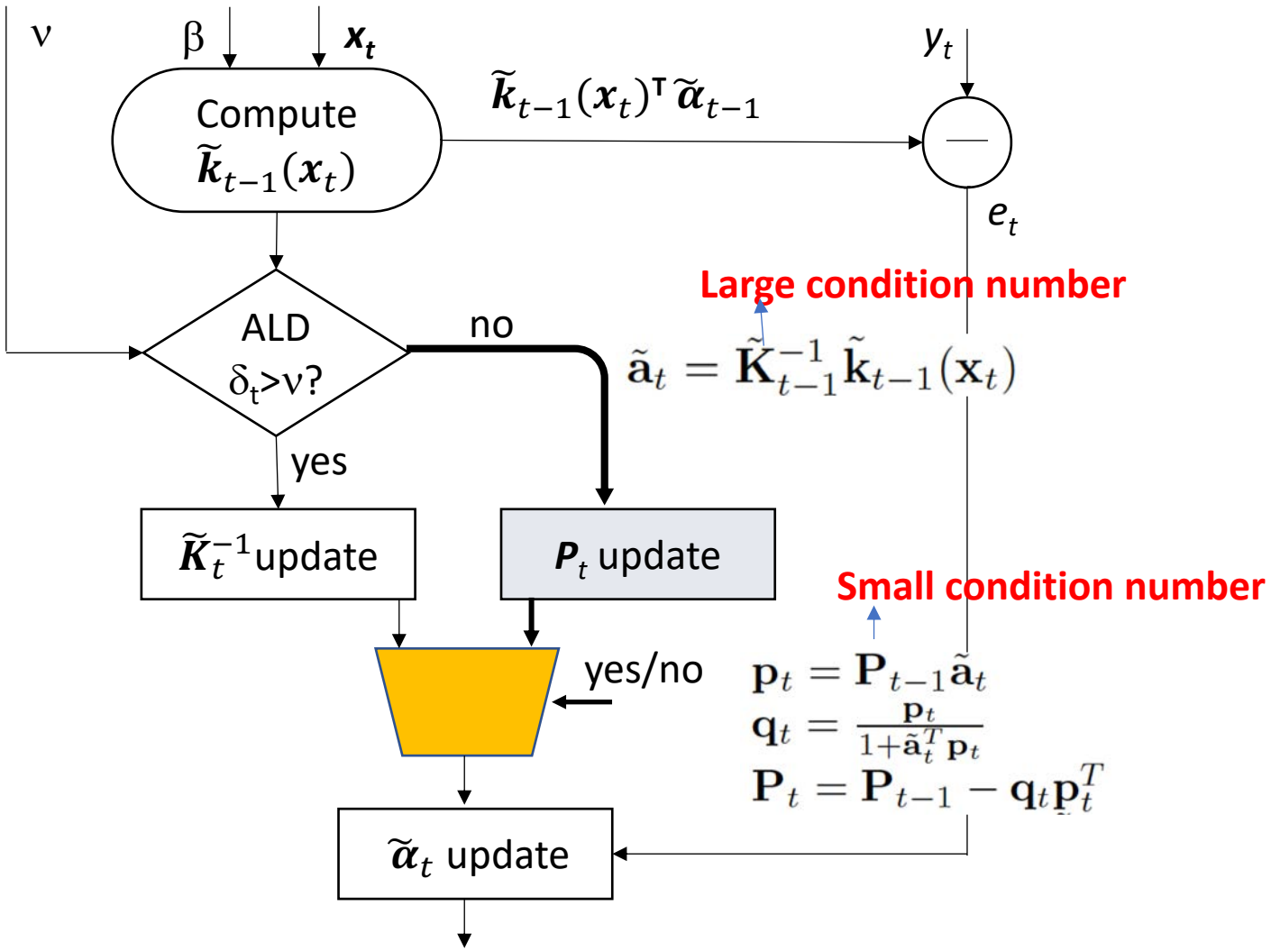
Case 3: Dot-product

$err_{rnd} \propto (|\mathbf{x}_1|^T|\mathbf{x}_2|)/|\mathbf{x}_1{}^T\mathbf{x}_2|\epsilon_R$

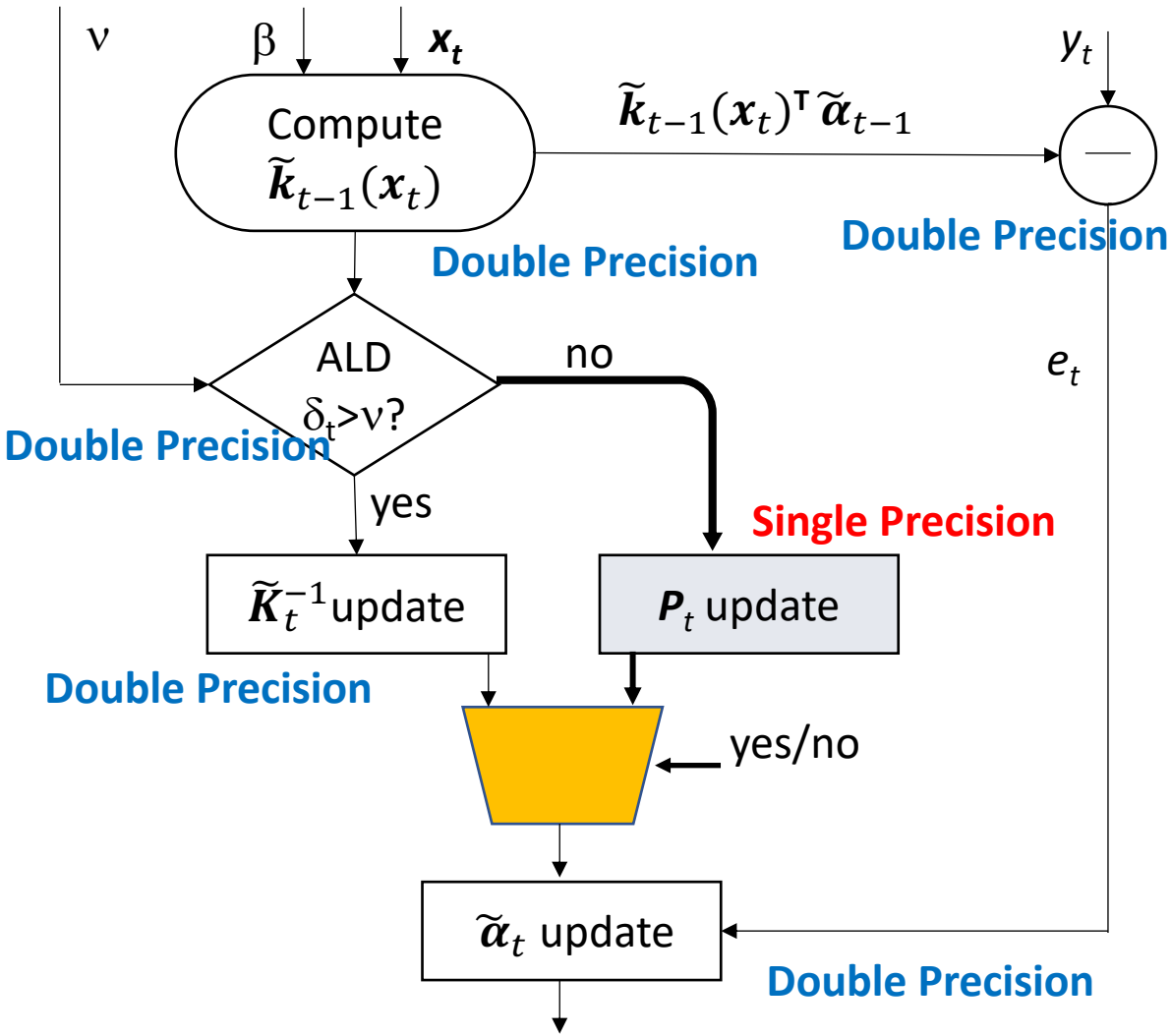
How can we use such properties?

Let us look at a Case Study with KRLS algorithm.

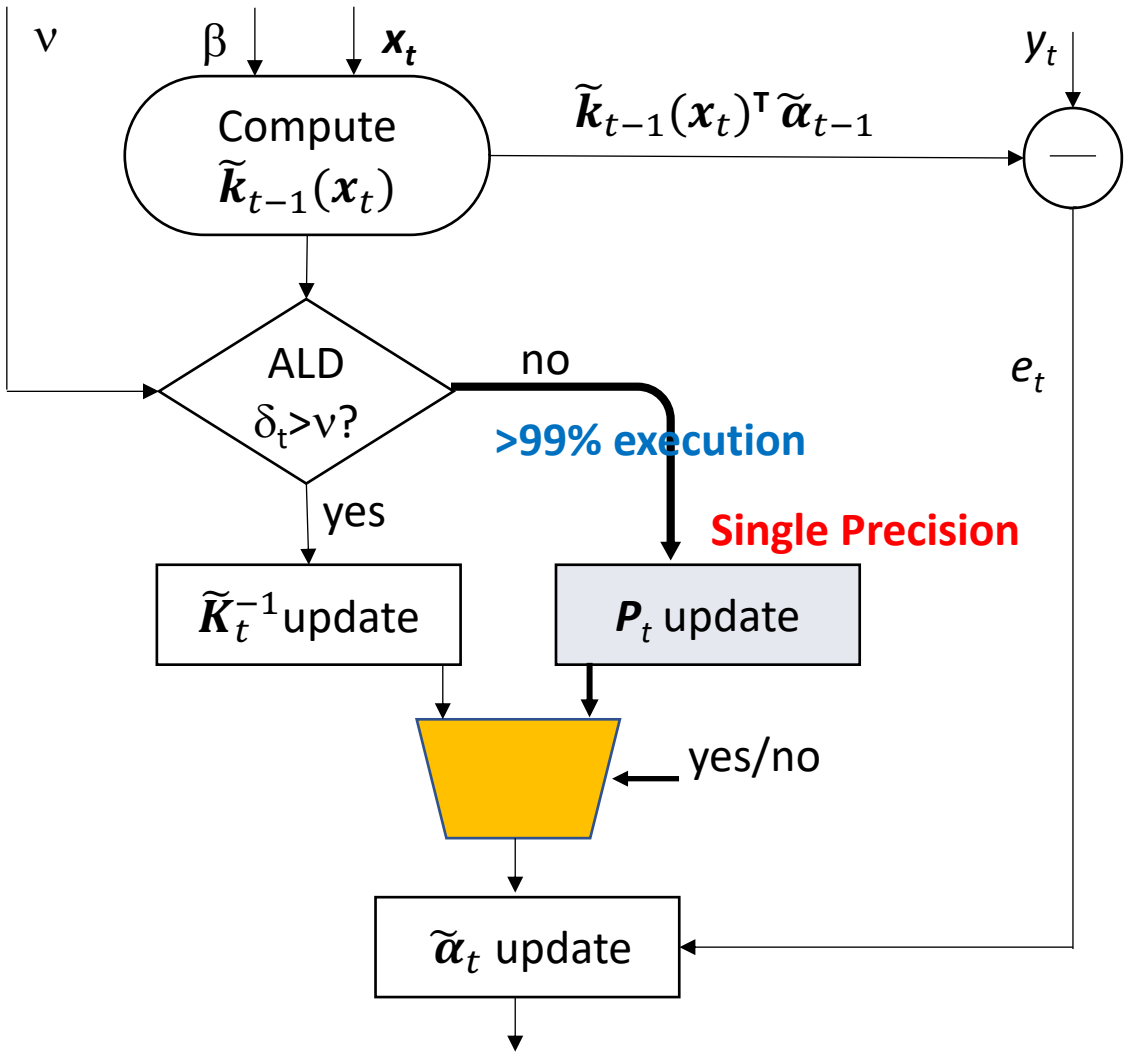
KRLS Algorithm Computing Components



Transprecision Computing for KRLS

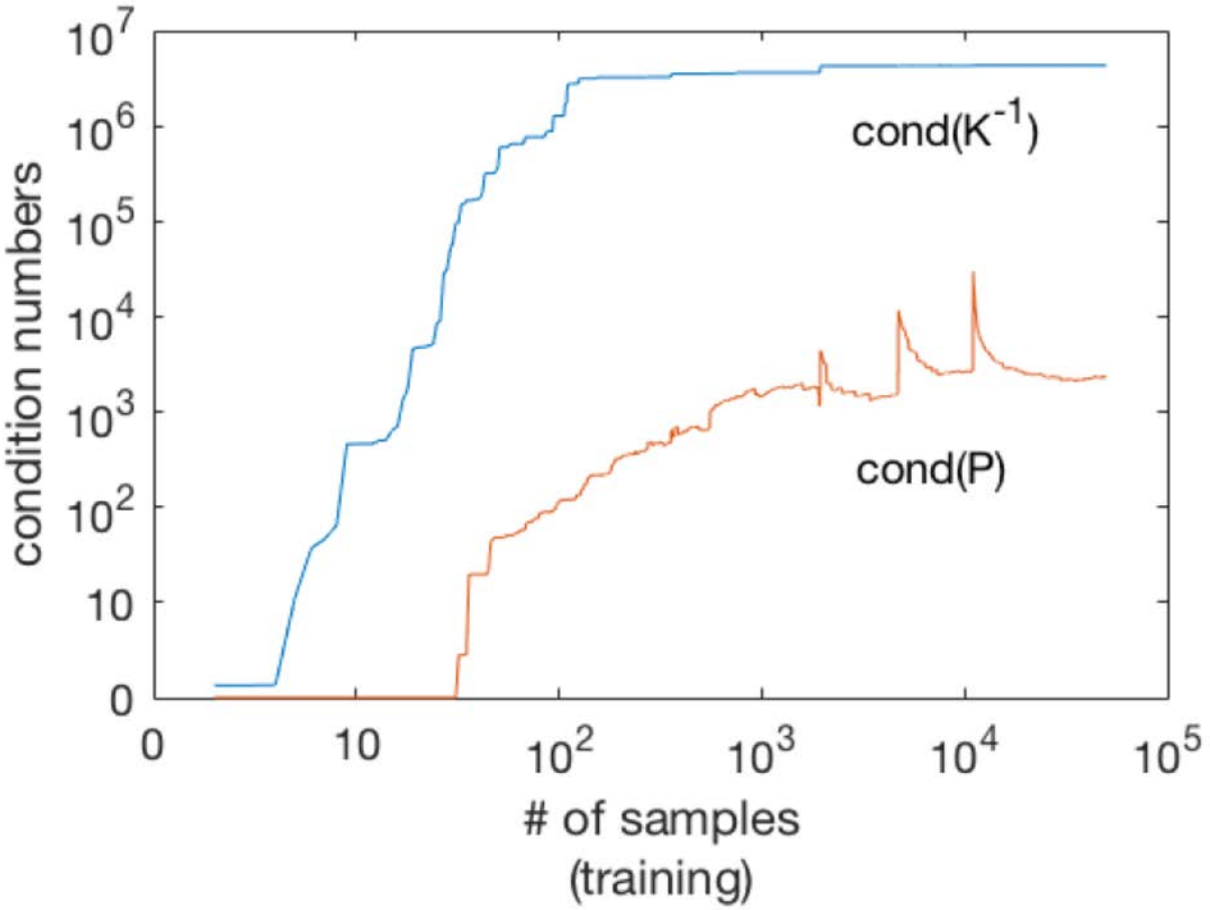


Transprecision Computing for KRLS



Condition Numbers of Matrices in KRLS

Condition number of K^{-1} and P with ALD ν and kernel width β from Cross-validation



Impact of Transprecision KRLS

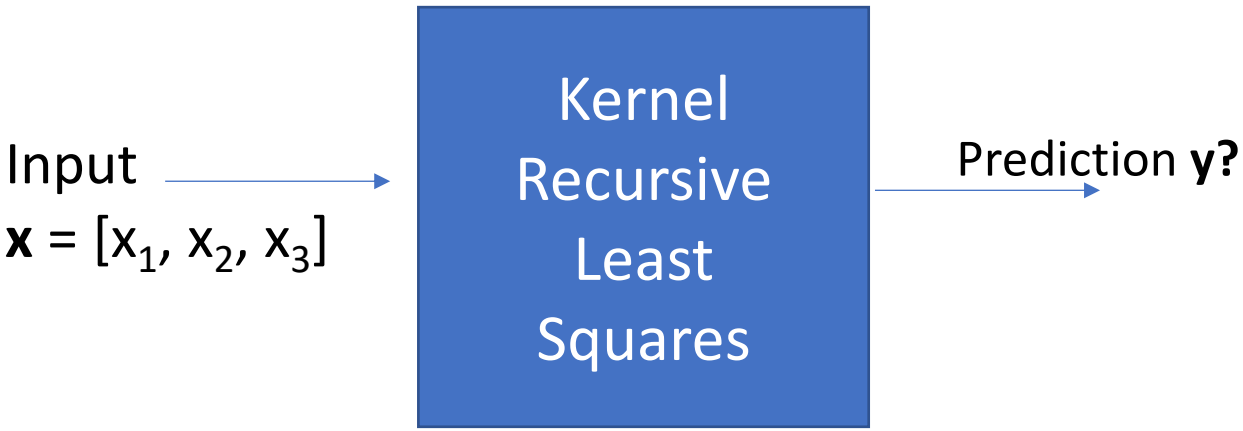
Non-Linear Regression Problem:

$$y = \sin(x_1)/x_1 + x_2/10.0 + \cos(x_3)$$

Impact of Transprecision KRLS

Non-Linear Regression Problem:

$$y = \sin(x_1)/x_1 + x_2/10.0 + \cos(x_3)$$



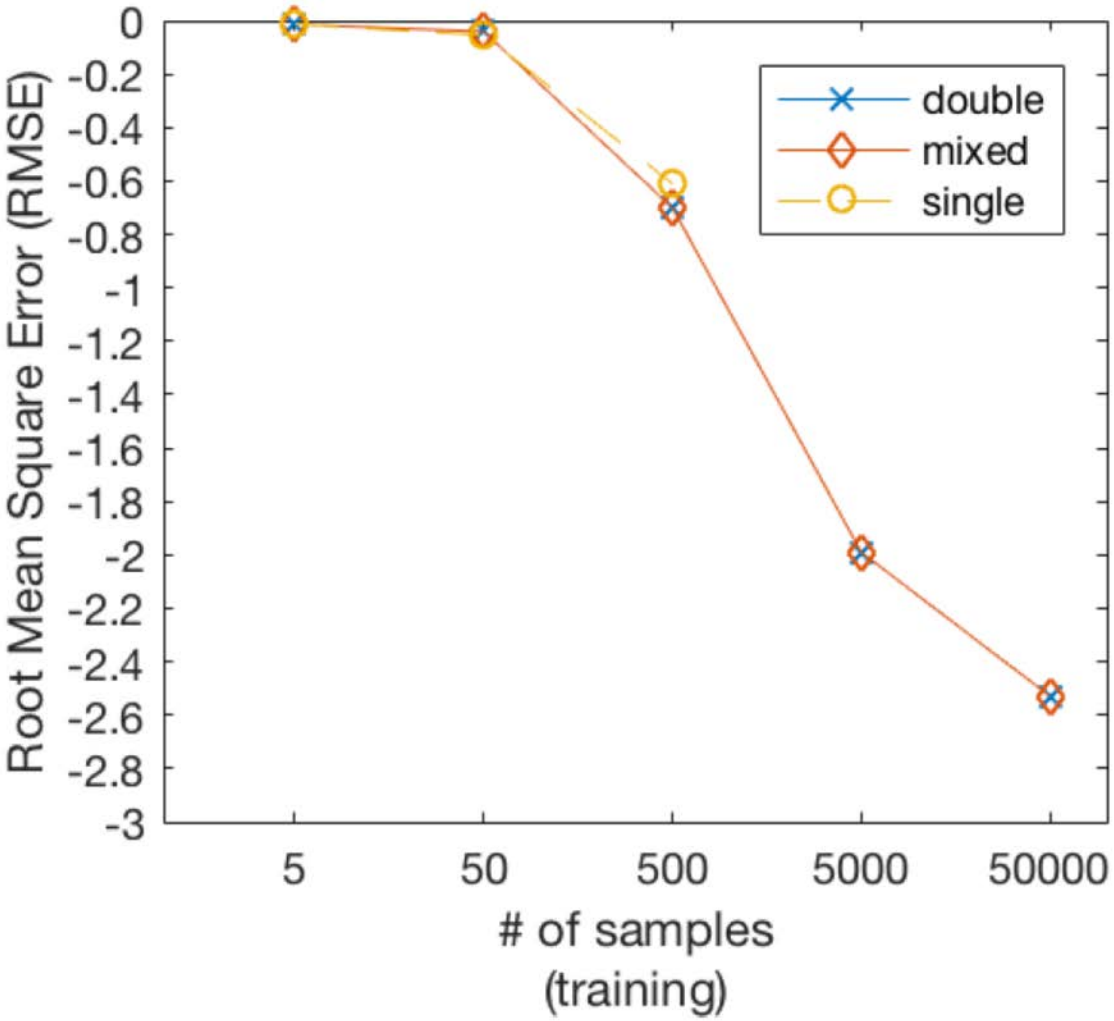
Impact of Transprecision KRLS

Non-Linear Regression Problem:

$$y = \sin(x_1)/x_1 + x_2/10.0 + \cos(x_3)$$

- x_1, x_2, x_3 = Uniform Random [-10, 10]
- Gaussian Kernel Width β = 2.5
- ALD ν = 0.01
- Intel(R) Xeon(R) CPU E5-2650 2GHz Single Core
- Energy Estimation: ALEA Energy Profiling Tool

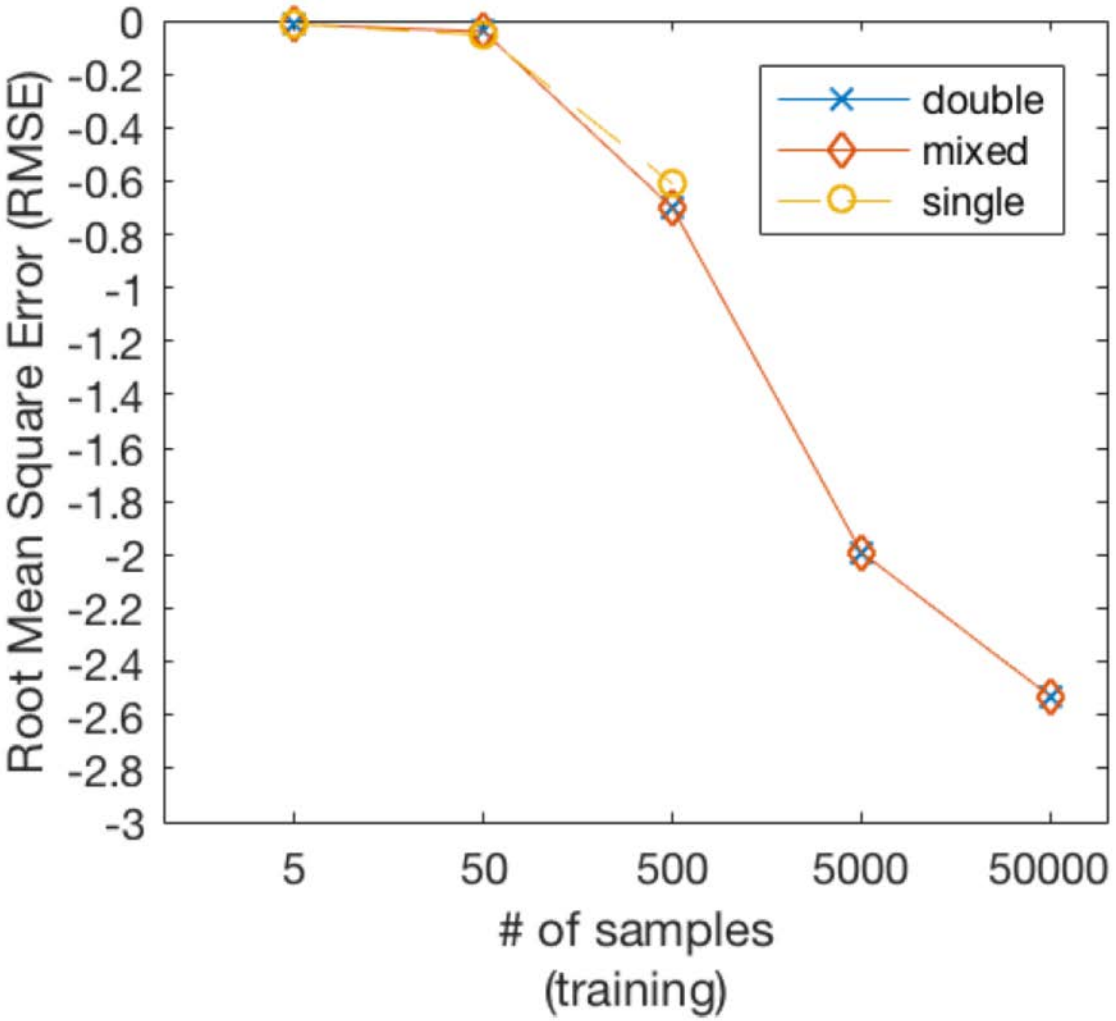
Prediction Accuracy



Almost 3 decimal digit accuracy
when # training data = 50,000

RMSE is used for accuracy measure

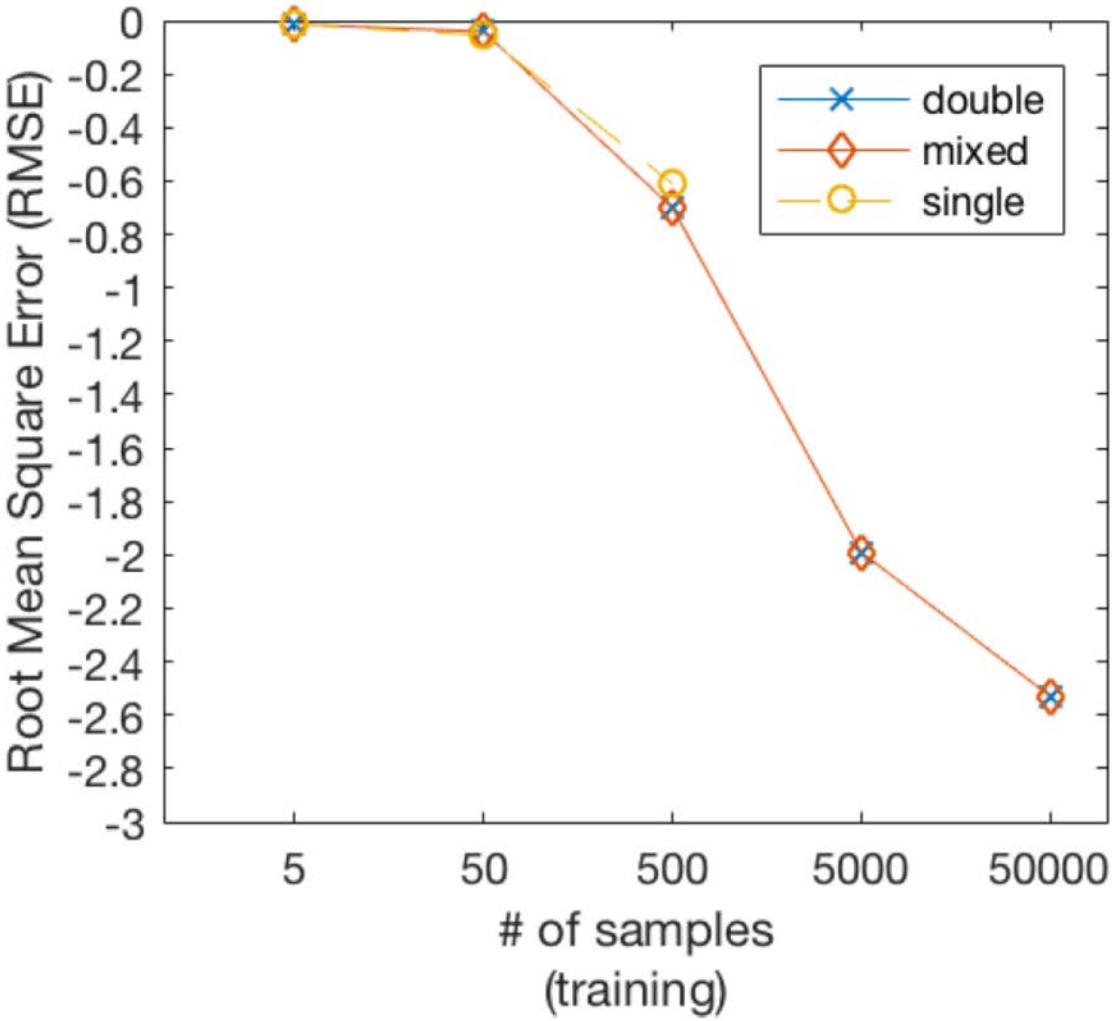
Prediction Accuracy



Almost 3 decimal digit accuracy
when # training data = 50,000

Entire Single Precision breaks KRLS

Prediction Accuracy

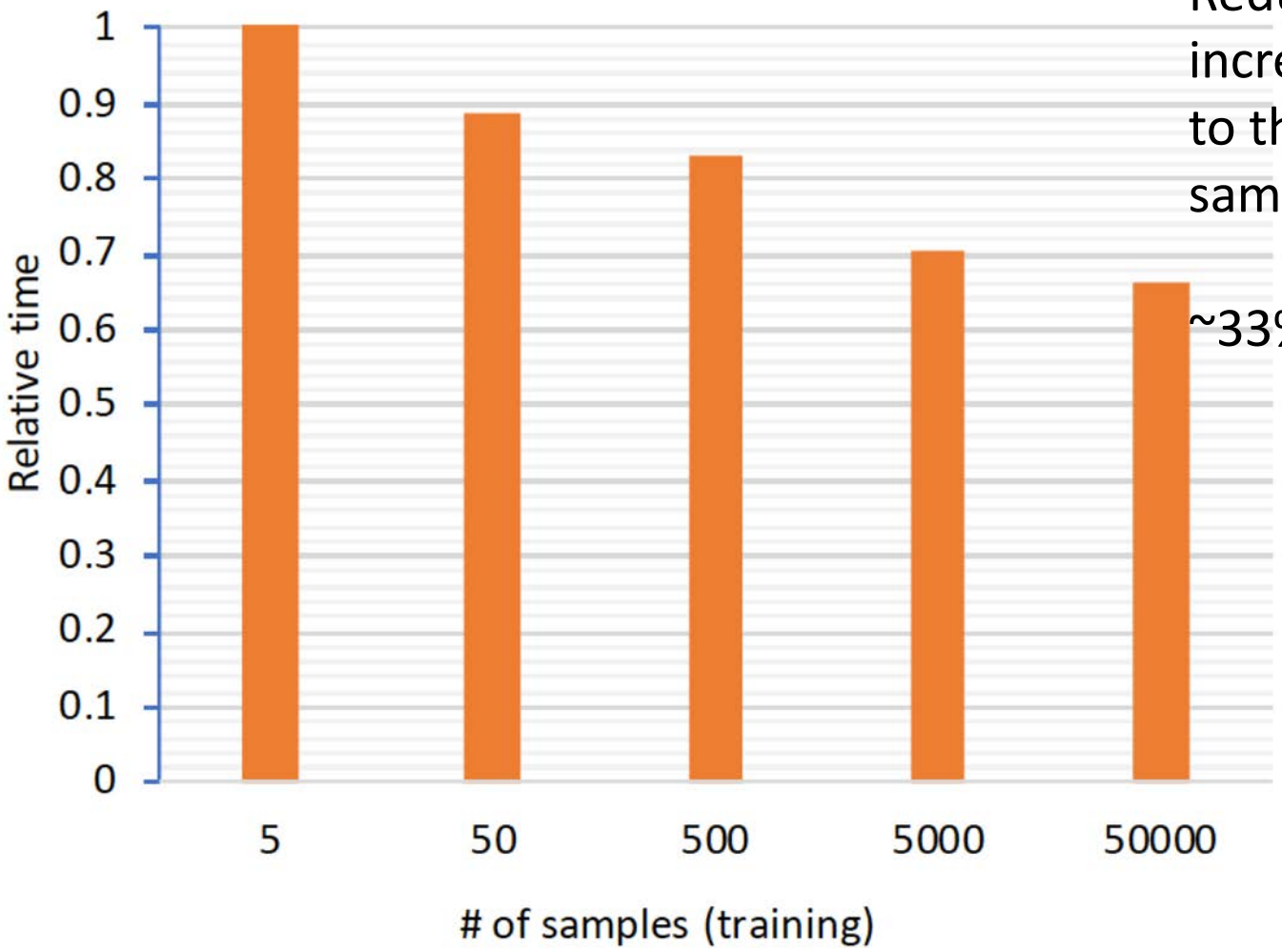


Almost 3 decimal digit accuracy
when # training data = 50,000

Entire Single Precision breaks KRLS

Identical Accuracy between double precision and transprecision KRLS

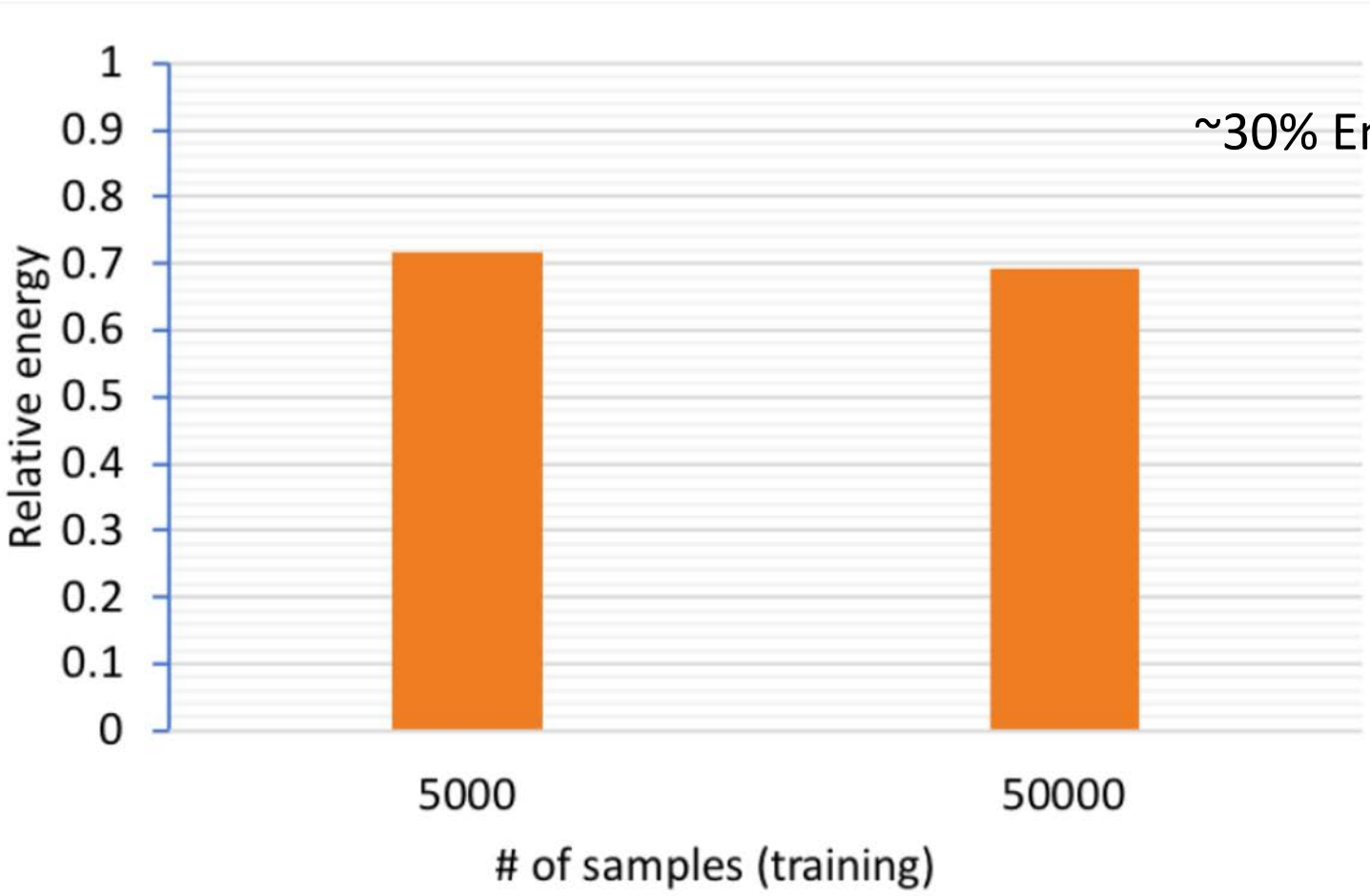
Execution Time Reduction by Transprecision KRLS



Reduction portion increases in proportion to the number of training samples

~33% Runtime Reduction

Energy Reduction by Transprecision KRLS



~30% Energy Reduction

Conclusions

Transprecision Computing:
Minimising Execution Time without accuracy loss and HW resource increment

Transprecision Techniques brought Significant Speedups and Energy Savings to Linear Solvers and KRLS without Accuracy Loss compared to Full precision arithmetic

Transprecision Computing can be a crucial paradigm for ManyCore in order to achieves both Speedups and Energy Savings.

Acknowledgements

: Energy Efficient Transprecision Techniques for Linear Solver Jun. 2018 – May 2020

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 798209 (Entrans).

: Open Transprecision Computing

Jan. 2017 – Dec. 2020

This project has received funding by the European Commission Horizon 2020 research and innovation programme under grant agreement No. 732631(OPRECOMP)

Thank you very much

Any Questions?