Range Reduction Based on Pythagorean Triples for Trigonometric Function Evaluation

IEEE ASAP 2015 Conference

Hugues de Lassus Saint-Geniès David Defour Guillaume Revy

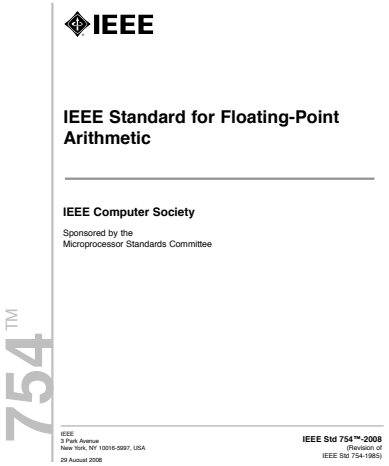
DALI, Université de Perpignan Via Domitia
LIRMM, Université de Montpellier
CNRS UMR 5506

July 27th, 2015

Context, objectives and achievements

Context and objectives

- IEEE-754 floating-point arithmetic


IEEE

IEEE Standard for Floating-Point Arithmetic

IEEE Computer Society

Sponsored by the
Microprocessor Standards Committee

754™

IEEE
3 Park Avenue
New York, NY 10016-5997, USA
29 August 2008

IEEE Std 754™-2008
(Revision of
IEEE Std 754-1985)

Context, objectives and achievements

Context and objectives

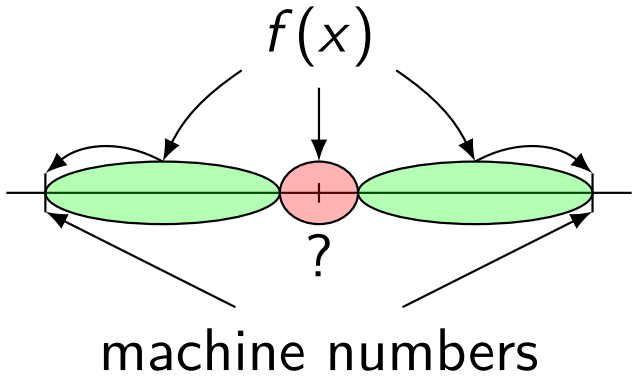
- IEEE-754 floating-point arithmetic
  - Elementary function evaluation

exp	log	expm1
sin	cos	arctan
$(1+x)^n$	x^y	...

Context, objectives and achievements

Context and objectives

- IEEE-754 floating-point arithmetic
  - Elementary function evaluation
  - Correct rounding



Context, objectives and achievements

Context and objectives

- IEEE-754 floating-point arithmetic
  - Elementary function evaluation
  - Correct rounding
- Trigonometric functions

exp	log	expm1
sin	cos	arctan
$(1+x)^n$	x^y	...

Context, objectives and achievements

Context and objectives

- IEEE-754 floating-point arithmetic
  - Elementary function evaluation
  - Correct rounding
- Trigonometric functions
- Table-based range reduction
  - Remove sources of error

exp	log	expm1
sin	cos	arctan
$(1+x)^n$	x^y	...

Context, objectives and achievements

Context and objectives

- IEEE-754 floating-point arithmetic
  - Elementary function evaluation
  - Correct rounding
- Trigonometric functions
- Table-based range reduction
  - Remove sources of error

exp	log	expm1
sin	cos	arctan
$(1+x)^n$	x^y	...

Achievements

- Algorithm to tabulate exact values for the sine and cosine functions

Context, objectives and achievements

Context and objectives

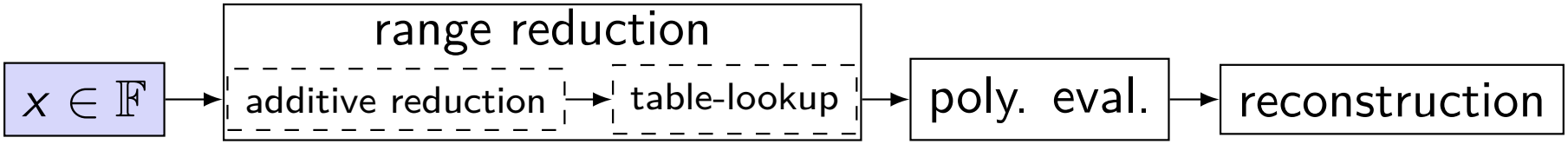
- IEEE-754 floating-point arithmetic
  - Elementary function evaluation
  - Correct rounding
- Trigonometric functions
- Table-based range reduction
  - Remove sources of error

exp	log	expm1
sin	cos	arctan
$(1+x)^n$	x^y	...

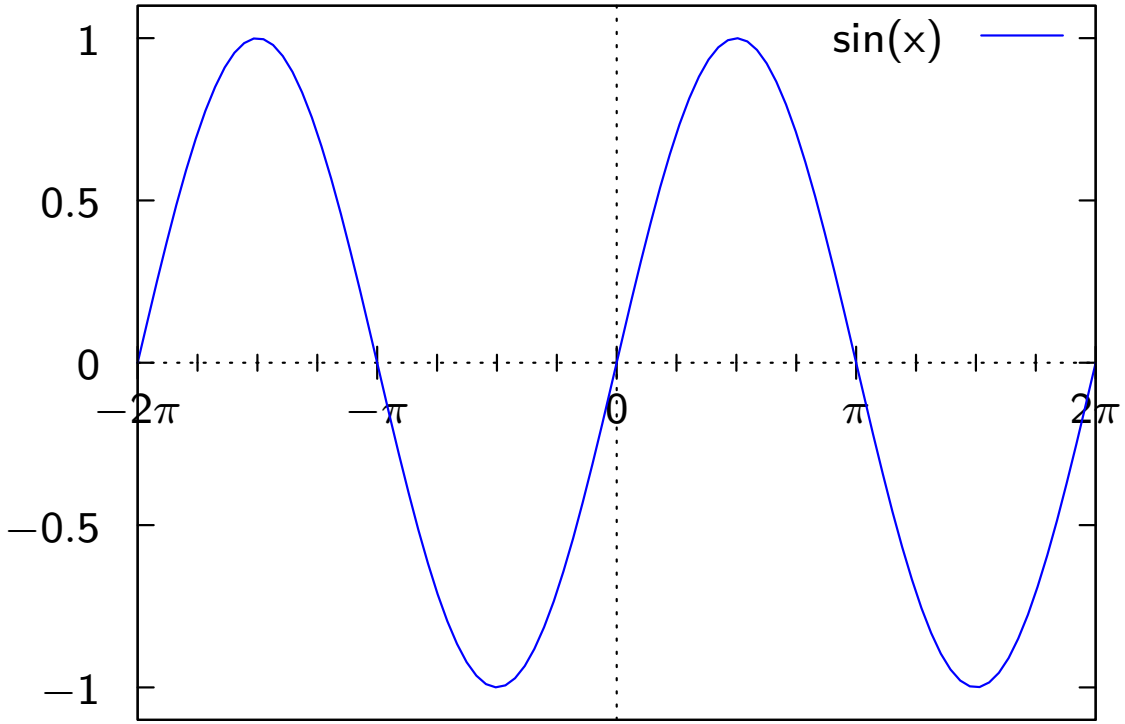
Achievements

- Algorithm to tabulate exact values for the sine and cosine functions
- For correct-rounding in double precision:
  - Precomputed table size reduced up to 45%
  - Memory accesses and FLOPs reduced up to 45% during the reconstruction

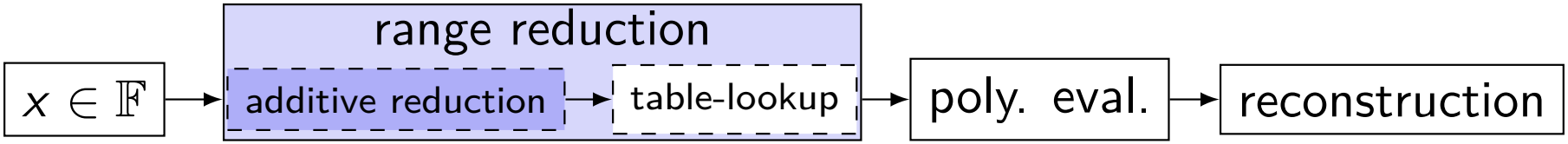
Additive range reduction



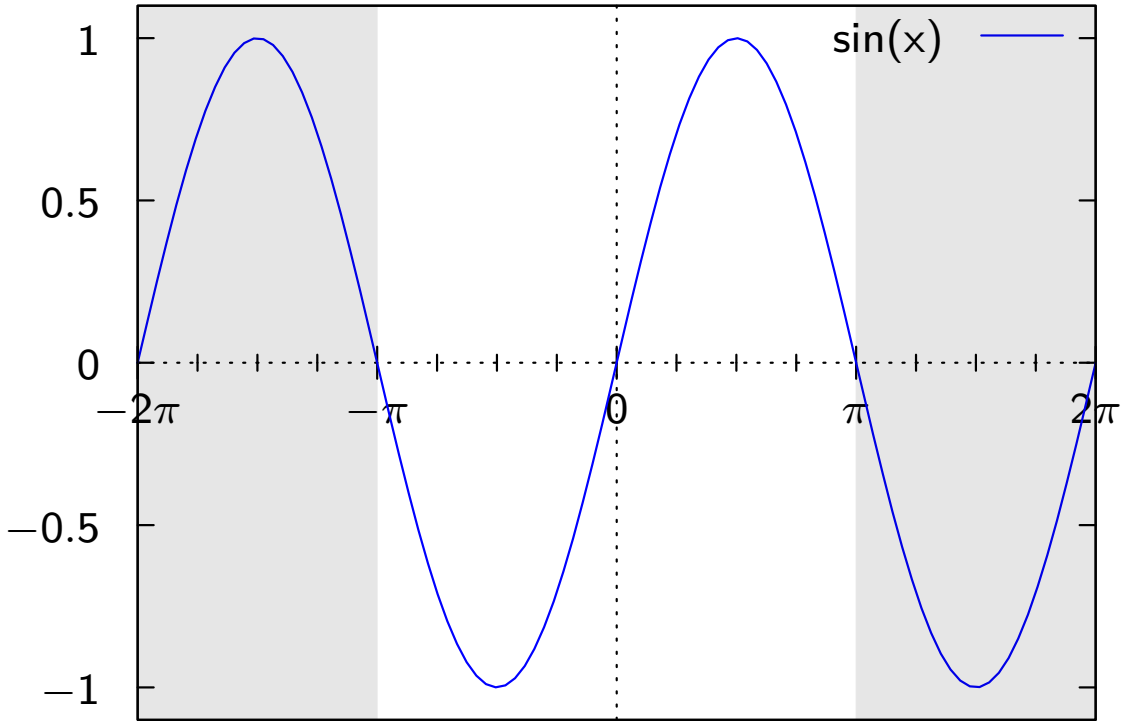
- Use of trigonometric **symmetries and periodicities**



Additive range reduction

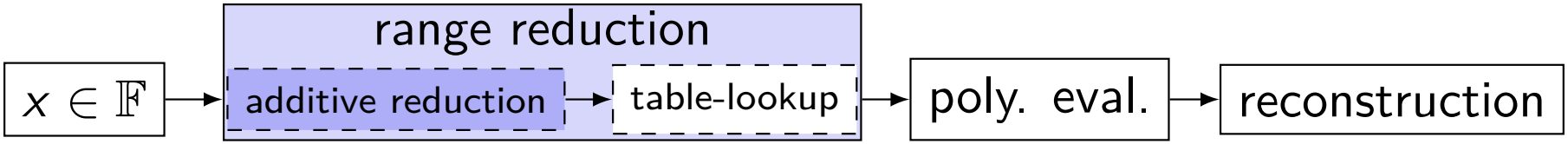


- Use of trigonometric **symmetries and periodicities**

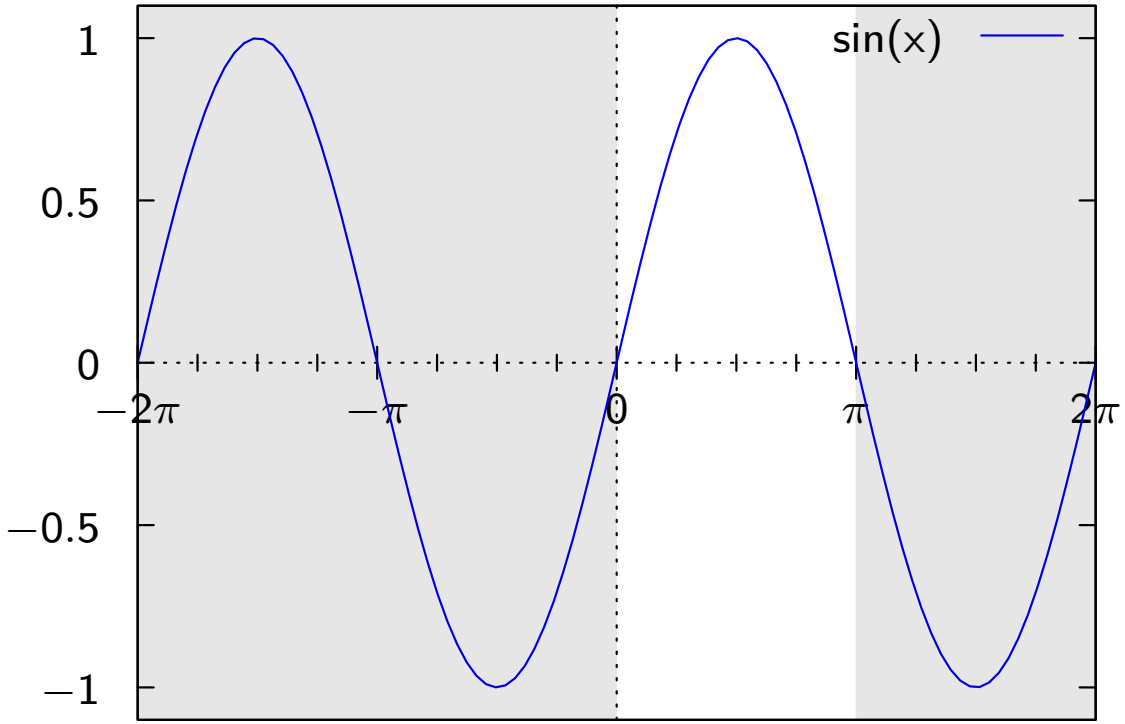


- $\sin(x) = \sin(x + 2\pi)$

Additive range reduction

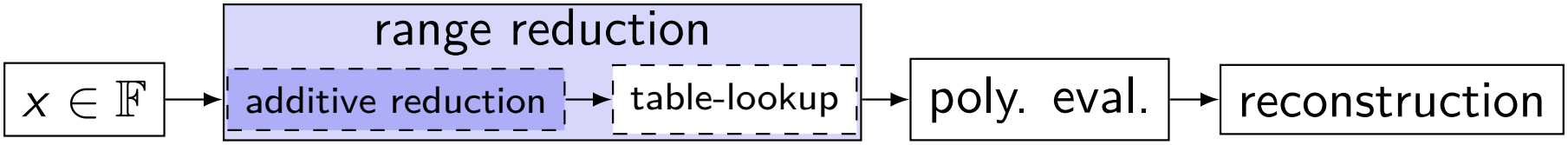


- Use of trigonometric **symmetries and periodicities**

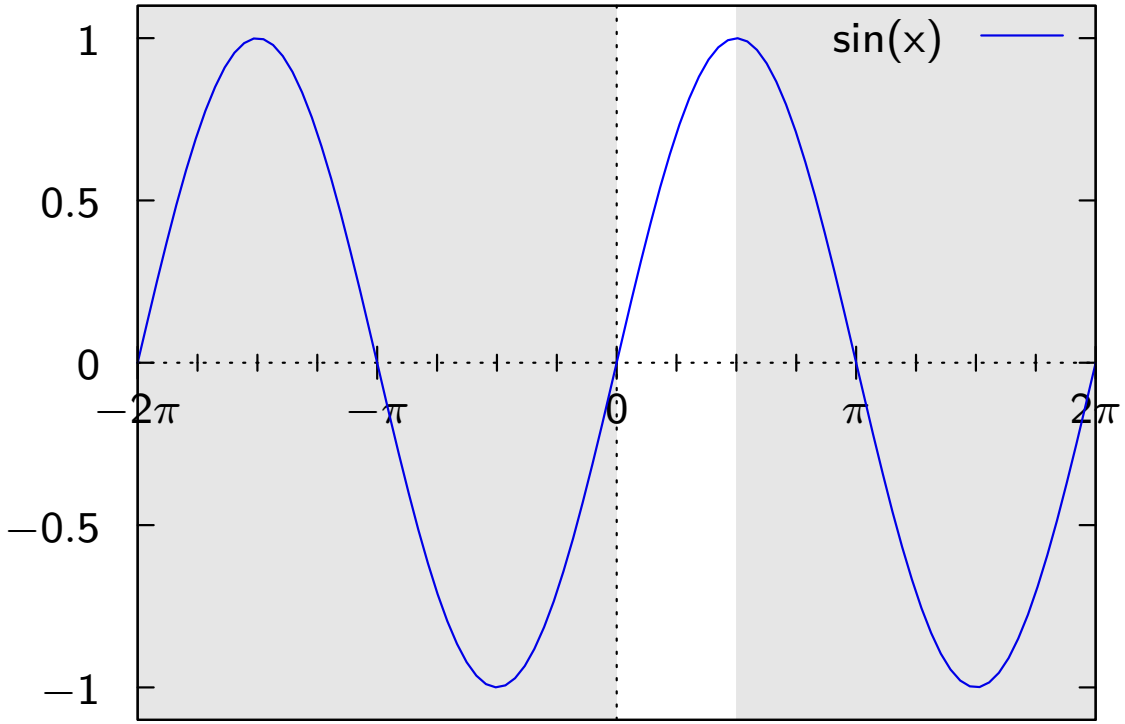


- $\sin(-x) = -\sin(x)$

Additive range reduction

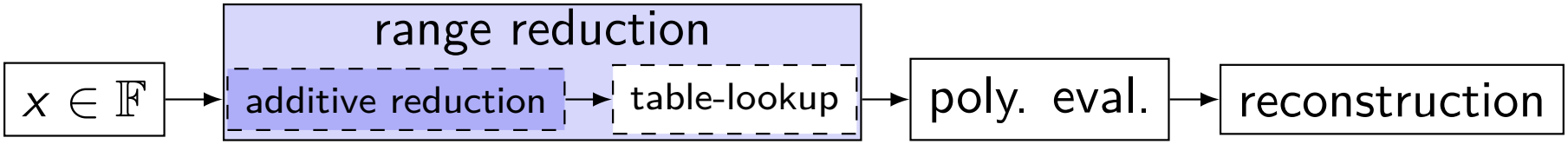


- Use of trigonometric **symmetries and periodicities**

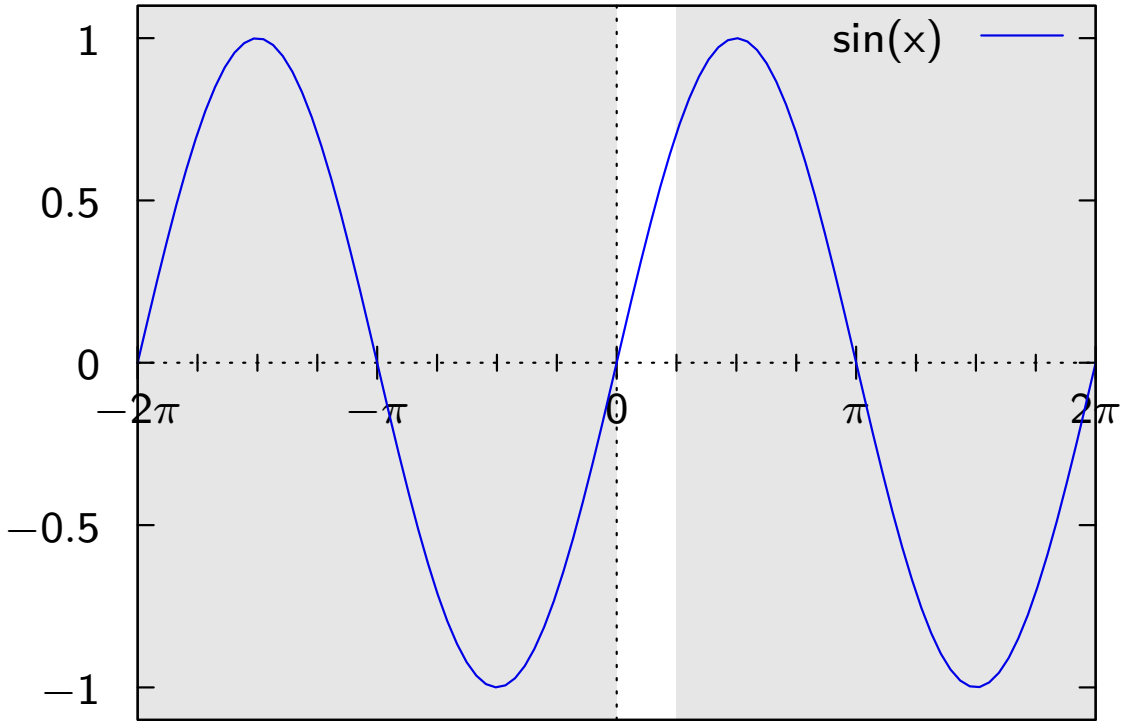


- $\sin(x) = \pm f_k\left(x - k \cdot \frac{\pi}{2}\right)$ with $f_k \in \{\sin, \cos\}$

Additive range reduction

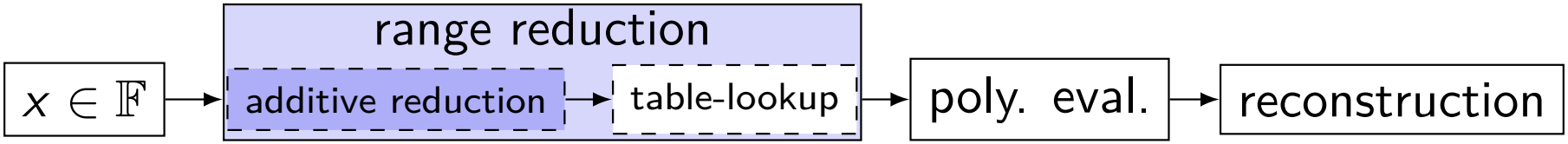


- Use of trigonometric **symmetries and periodicities**

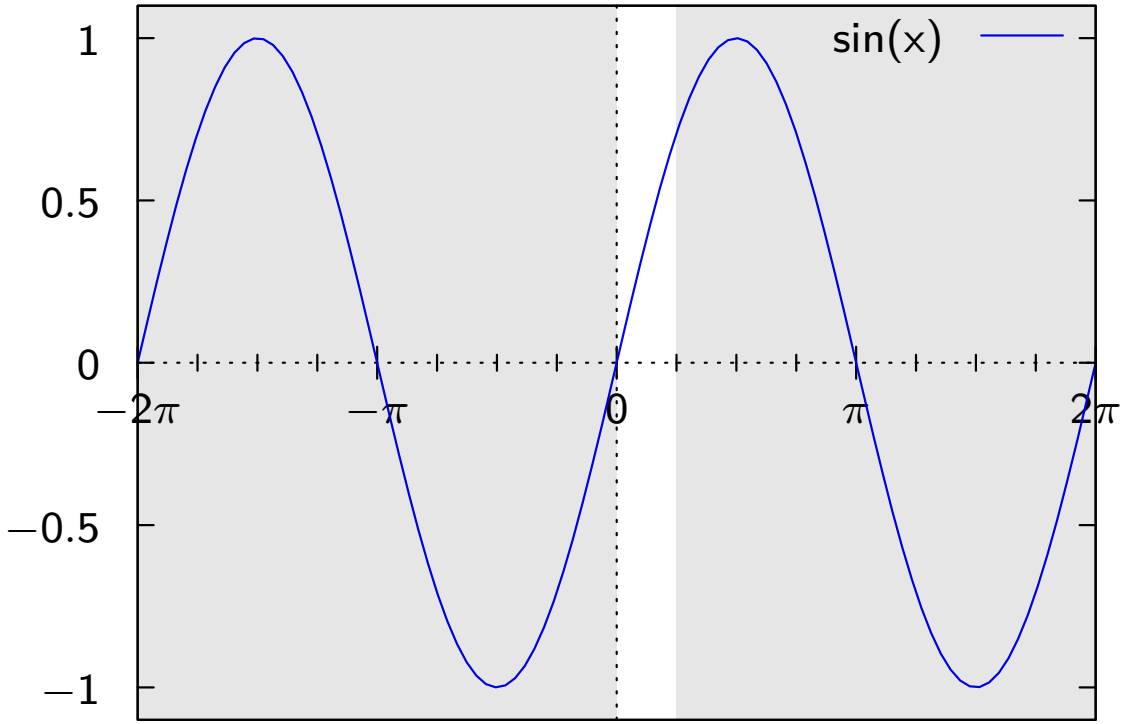


- $\sin(x) = \pm f_k\left(x - k \cdot \frac{\pi}{2}\right)$ with $f_k \in \{\sin, \cos\}$

Additive range reduction

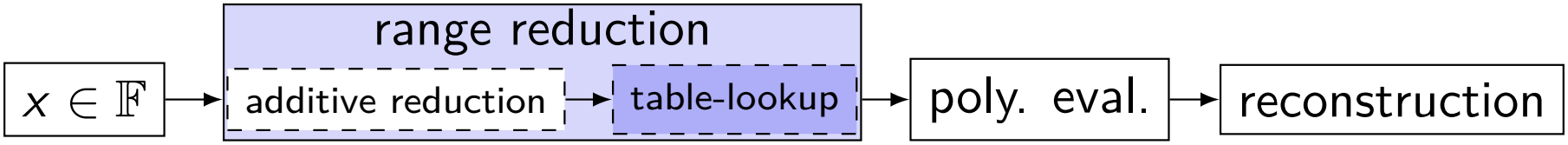


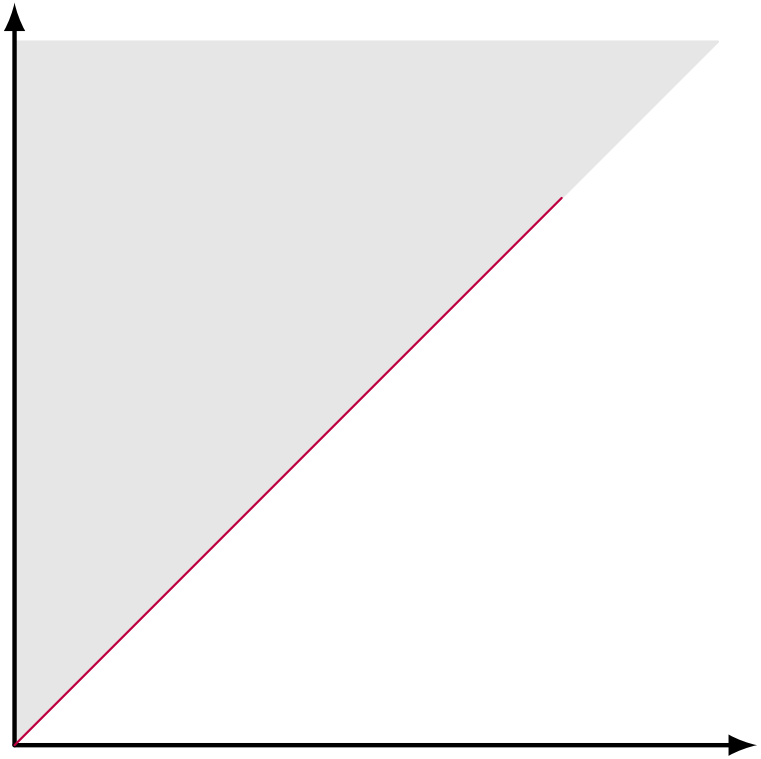
- Use of trigonometric **symmetries and periodicities**



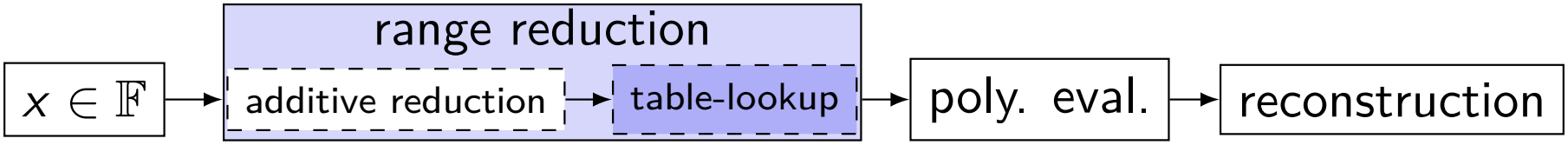
$\Rightarrow$ Range reduction $x \in \mathbb{F} \mapsto \mathbf{x}^* \in \left[\mathbf{0}, \frac{\pi}{4}\right]$

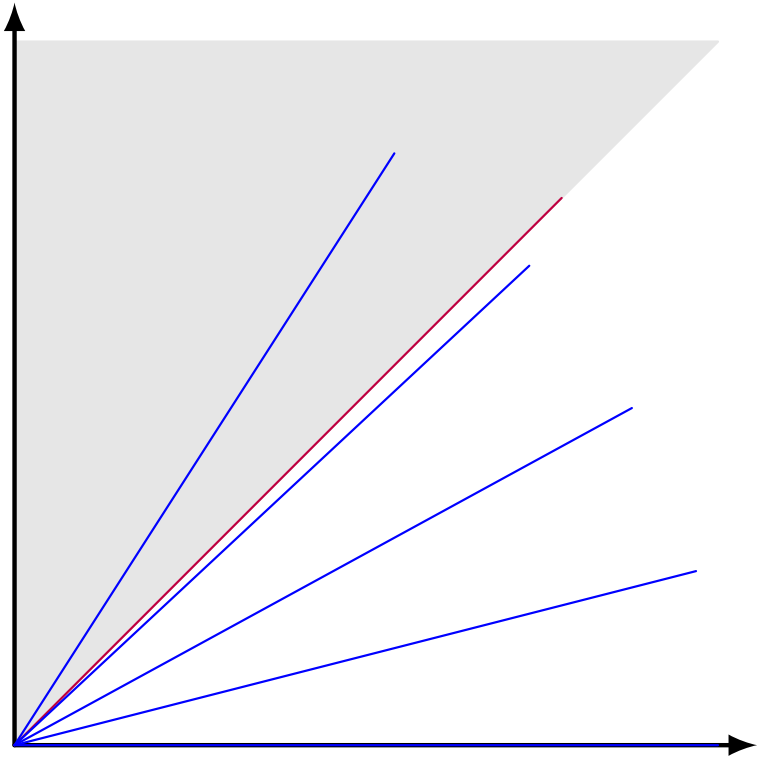
Tang's table-based range reduction





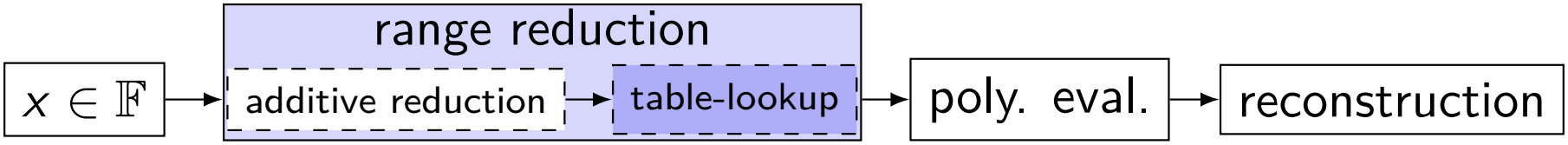
Tang's table-based range reduction

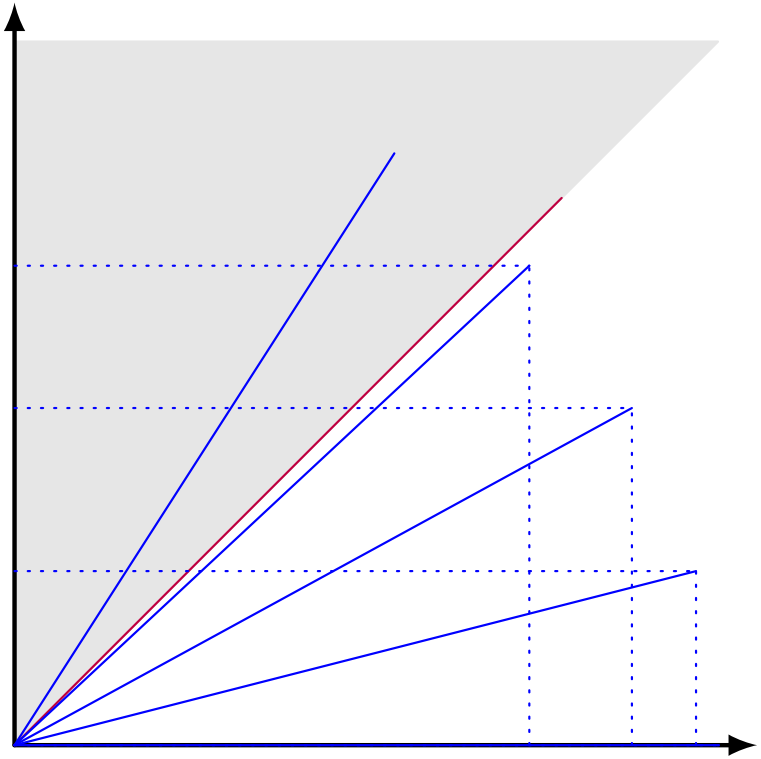




$\rightarrow$ regularly spaced angles
$x_h(i) = i \cdot 2^{-p}$

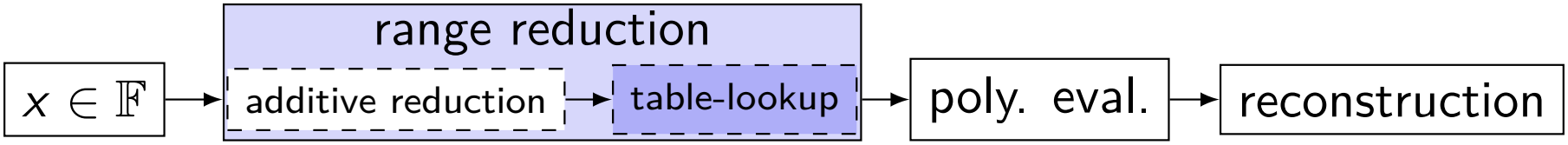
Tang's table-based range reduction

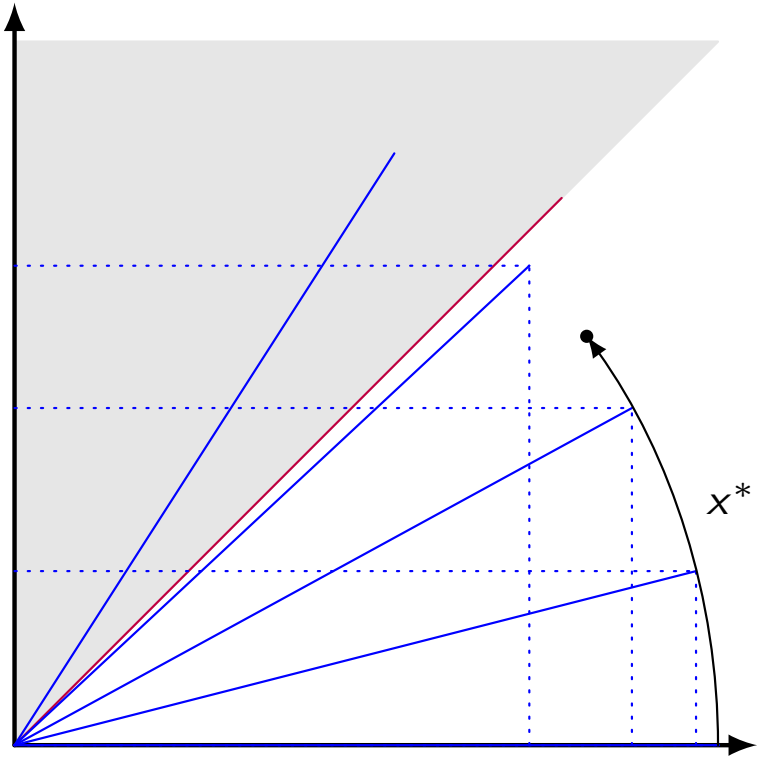




$\rightarrow$ regularly spaced angles
$x_h(i) = i \cdot 2^{-p}$

Tang's table-based range reduction

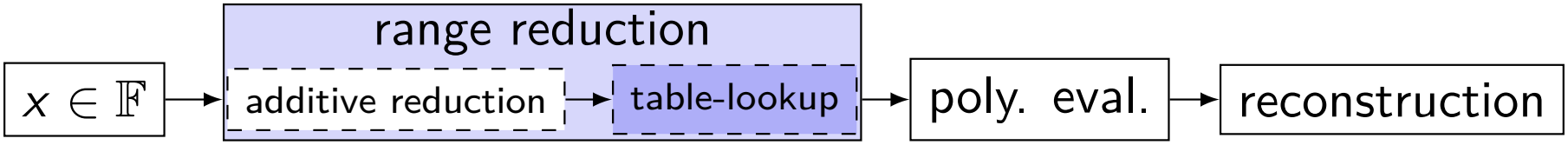


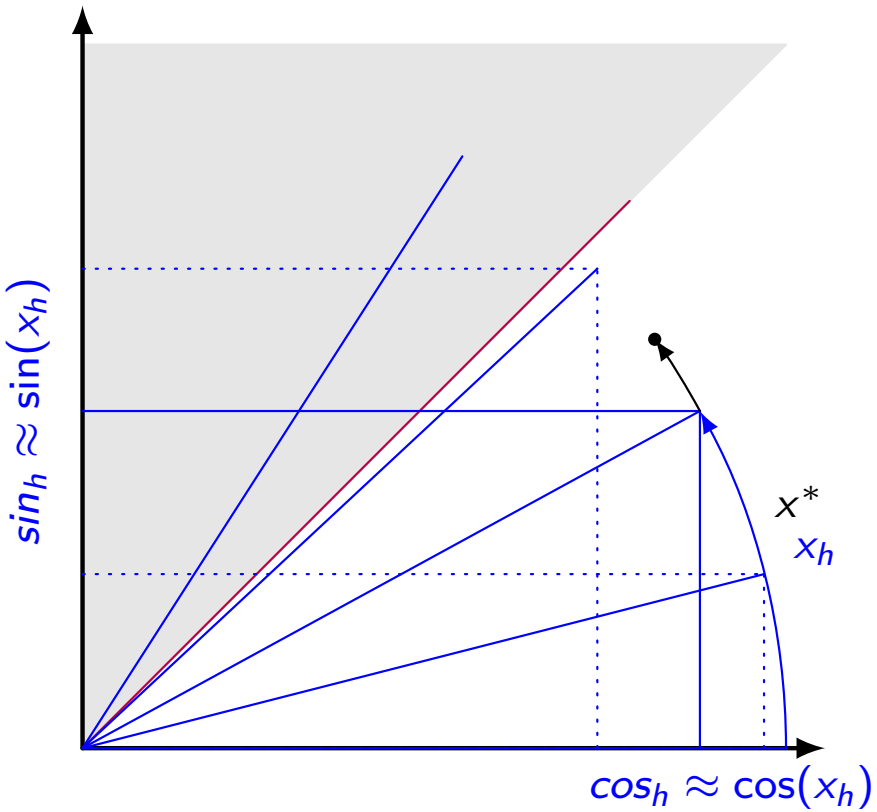


$\rightarrow$ regularly spaced angles
$x_h(i) = i \cdot 2^{-p}$

$$x^* = \underbrace{x.xx \cdots xx}\ \underbrace{xx \cdots xx}$$

Tang's table-based range reduction

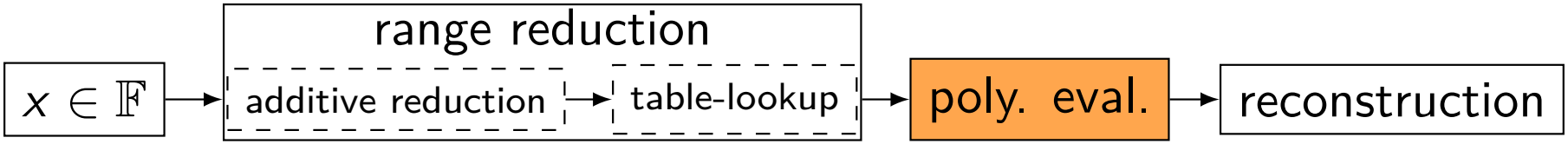


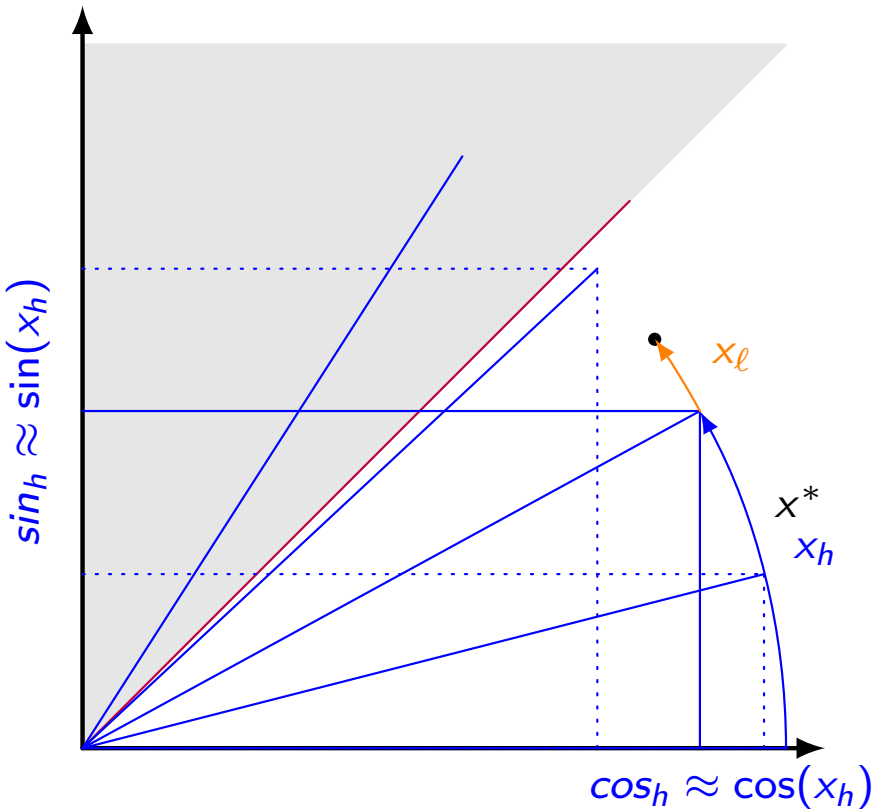


$\rightarrow$ regularly spaced angles
$x_h(i) = i \cdot 2^{-p}$

$$x^* = \underbrace{x.xx \cdots xx}_{p \text{ bits: } x_h} \underbrace{xx \cdots xx}$$

Tang's table-based range reduction

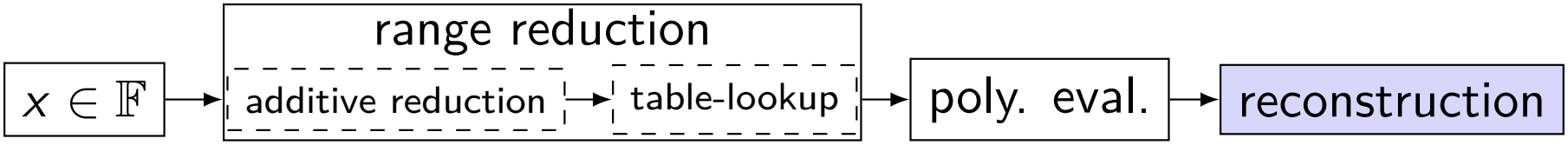


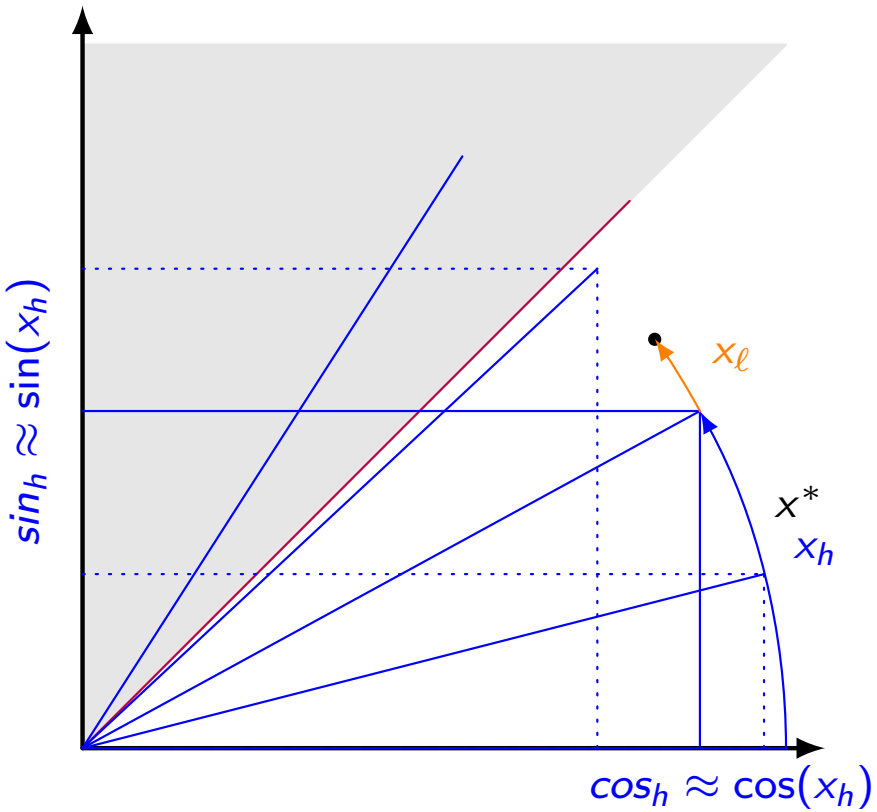


$\rightarrow$ regularly spaced angles
$x_h(i) = i \cdot 2^{-p}$

$$x^* = \underbrace{x.xx \cdots xx}_{p \text{ bits: } x_h} \underbrace{xx \cdots xx}_{x_\ell}$$

Tang's table-based range reduction



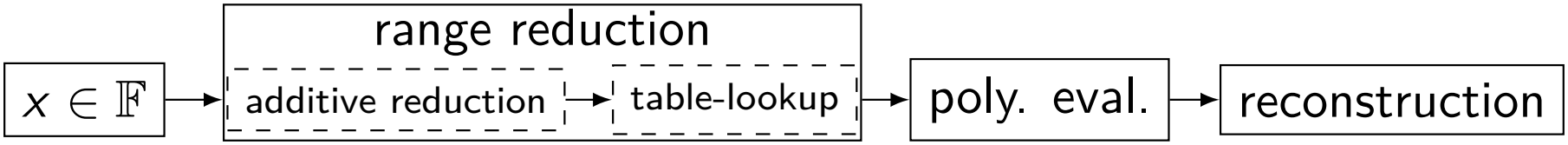


$\rightarrow$ regularly spaced angles
$x_h(i) = i \cdot 2^{-p}$

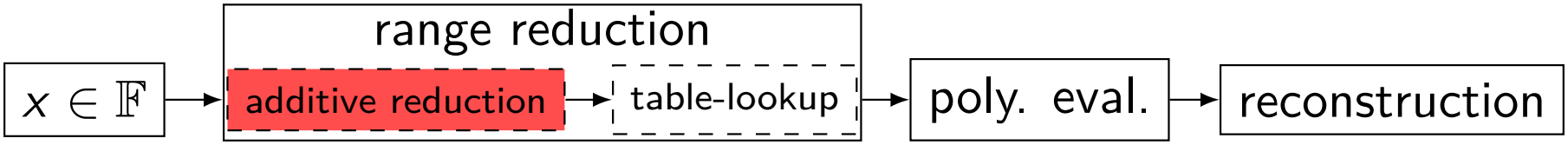
$$x^* = \underbrace{x.xx \cdots xx}_{p \text{ bits: } x_h} \underbrace{xx \cdots xx}_{x_\ell}$$

$$\sin(x^*) \approx sin_h \otimes \cos(x_\ell) \oplus cos_h \otimes \sin(x_\ell)$$

Sources of error in the classical method



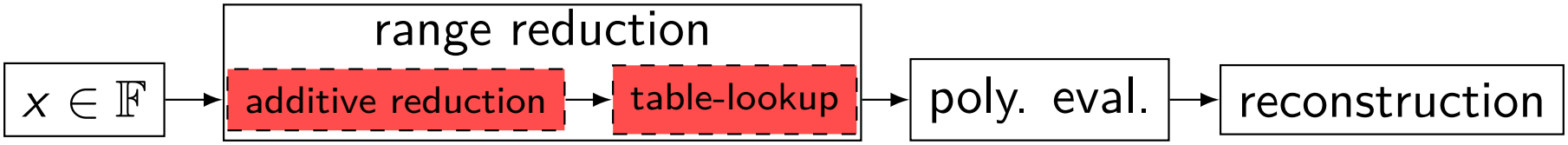
Sources of error in the classical method



Errors appear:

- in x^*, because of the additive range reduction

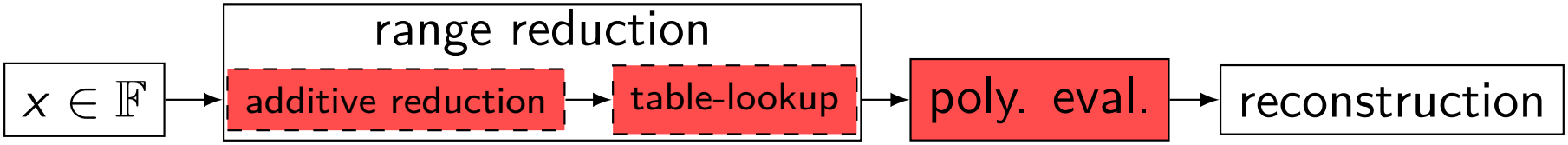
Sources of error in the classical method



Errors appear:

- in x^*, because of the additive range reduction
- in sin_h and cos_h, which are rounded approximations

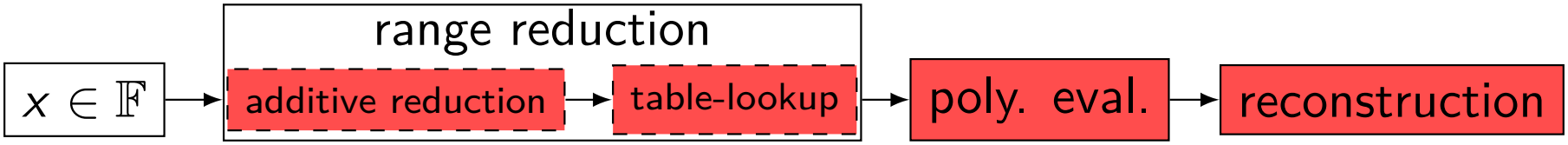
Sources of error in the classical method



Errors appear:

- in x^*, because of the additive range reduction
- in sin_h and cos_h, which are rounded approximations
- in $\sin(x_\ell)$ and $\cos(x_\ell)$, computed by polynomial evaluations

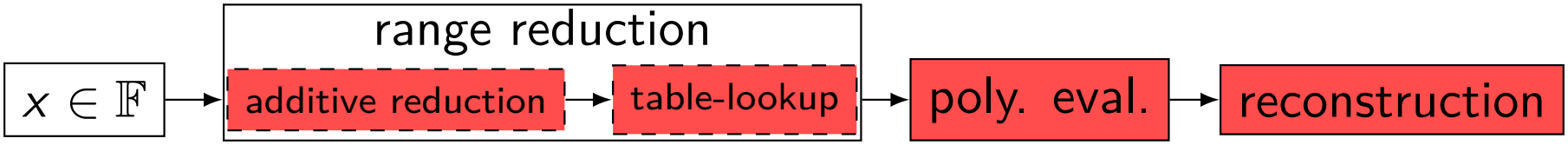
Sources of error in the classical method



Errors appear:

- in x^*, because of the additive range reduction
- in sin_h and cos_h, which are rounded approximations
- in $\sin(x_\ell)$ and $\cos(x_\ell)$, computed by polynomial evaluations
- during the reconstruction process

Sources of error in the classical method

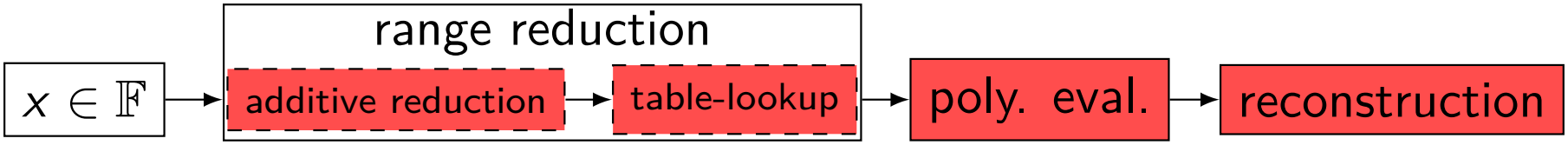


Errors appear:

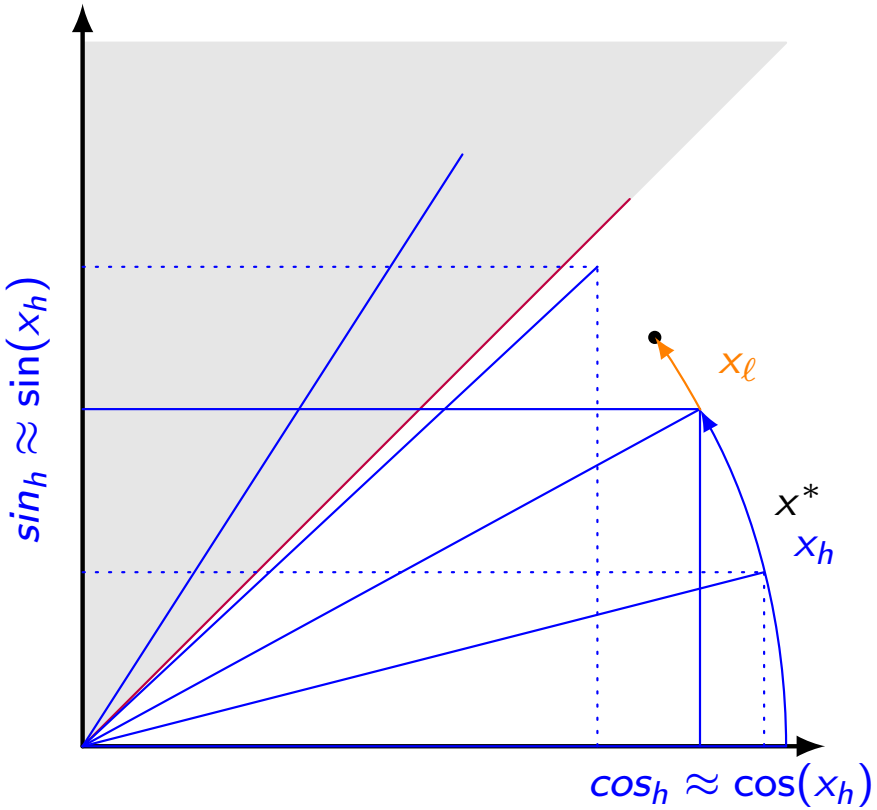
- in x^*, because of the additive range reduction
- in sin_h and cos_h, which are rounded approximations
- in $\sin(x_\ell)$ and $\cos(x_\ell)$, computed by polynomial evaluations
- during the reconstruction process

Drawback: additional bits in computation steps for accuracy purposes

An improvement: Gal's accurate tables



- Small perturbations to the table inputs $\rightarrow$ *corr*
- sin_h **and** cos_h 10 to 21 bits **closer to** machine numbers

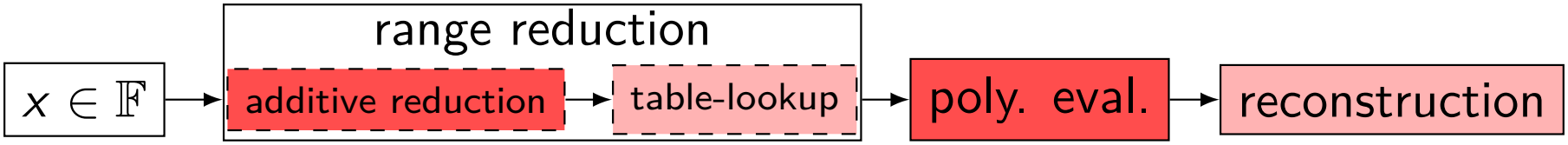


$\sin(x_h)$ and $\cos(x_h)$ look like

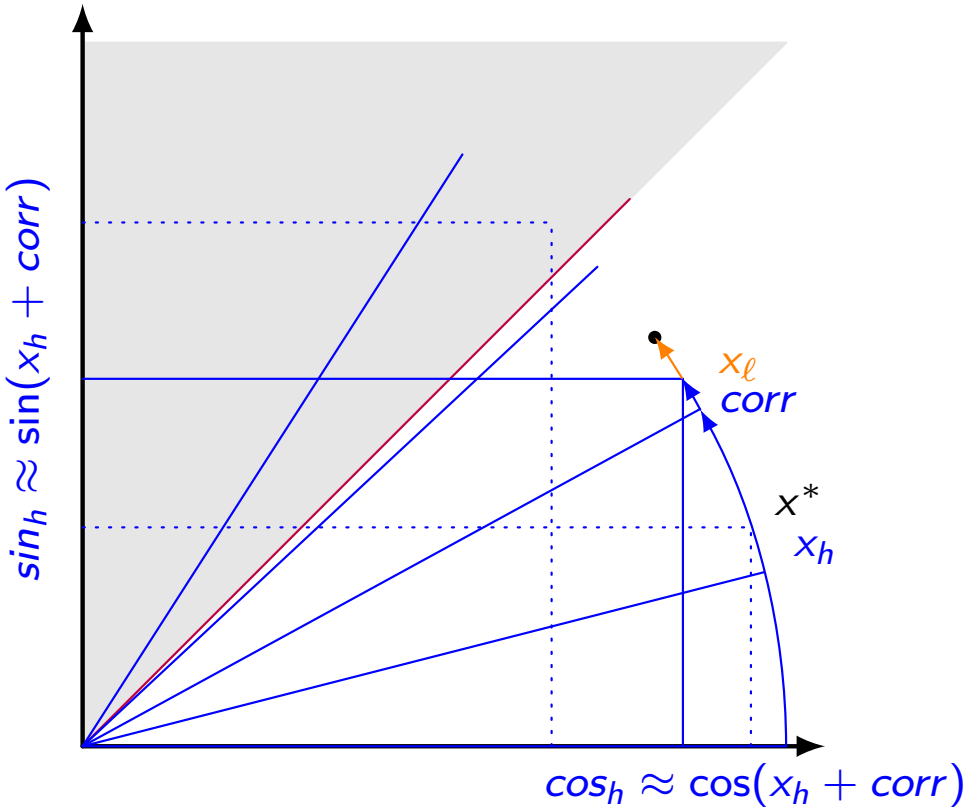
$$\underbrace{X.XX\cdots XX}_{n \text{ tabulated bits}}\underbrace{0111010111001\cdots}_{\text{“random” bits}}$$

$$\sin(x^*) \approx sin_h \otimes \cos(x_\ell) \oplus cos_h \otimes \sin(x_\ell)$$

An improvement: Gal's accurate tables



- Small perturbations to the table inputs $\rightarrow$ *corr*
- sin_h **and** cos_h 10 to 21 bits **closer to** machine numbers

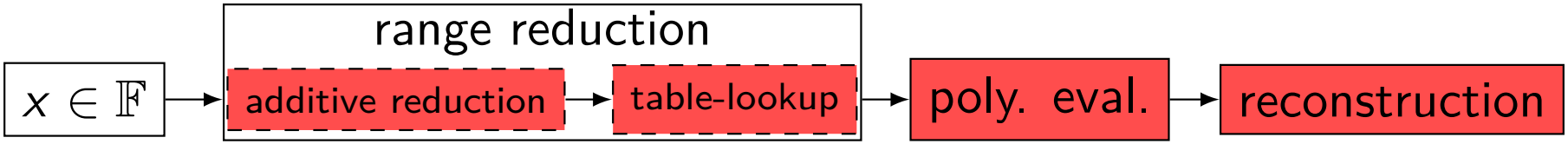


$\sin(x_h)$ and $\cos(x_h)$ look like

$$\underbrace{X.XX\cdots XX}_{n \text{ tabulated bits}}\ \underbrace{00\cdots 00}_{10 \text{ to } 21 \text{ bits}}\ 10101\cdots$$

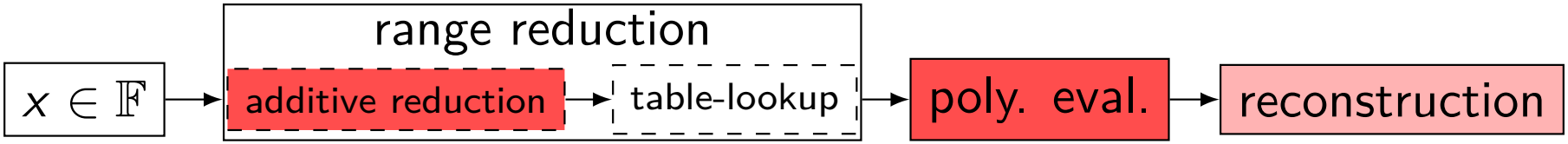
$$\sin(x^*) \approx sin_h \otimes \cos(x_\ell - corr) \oplus cos_h \otimes \sin(x_\ell - corr)$$

Key idea: remove the error on tabulated values



Our conception of **how error must be treated:**

Key idea: remove the error on tabulated values

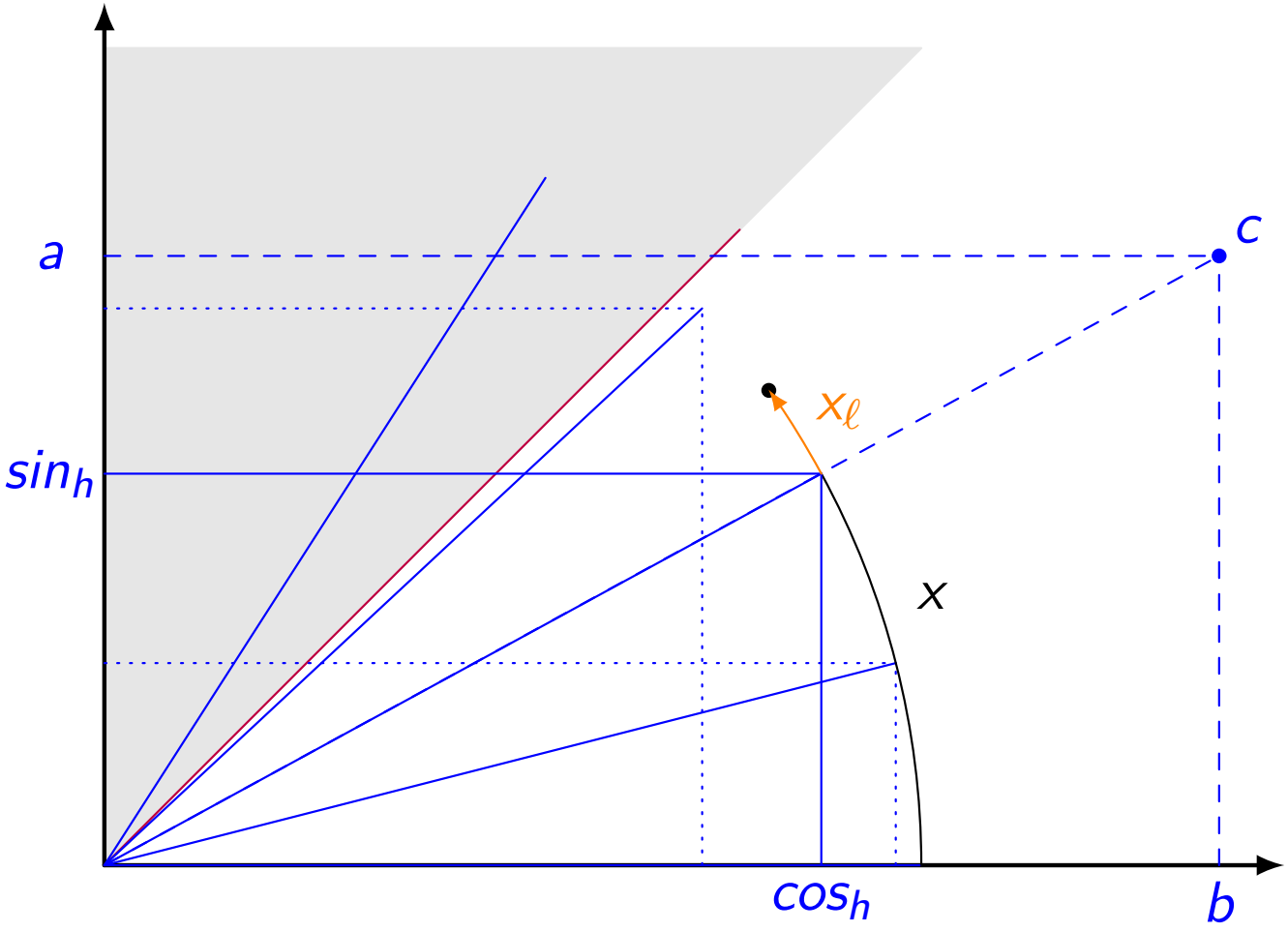


Our conception of **how error must be treated:**

remove it whenever you can!

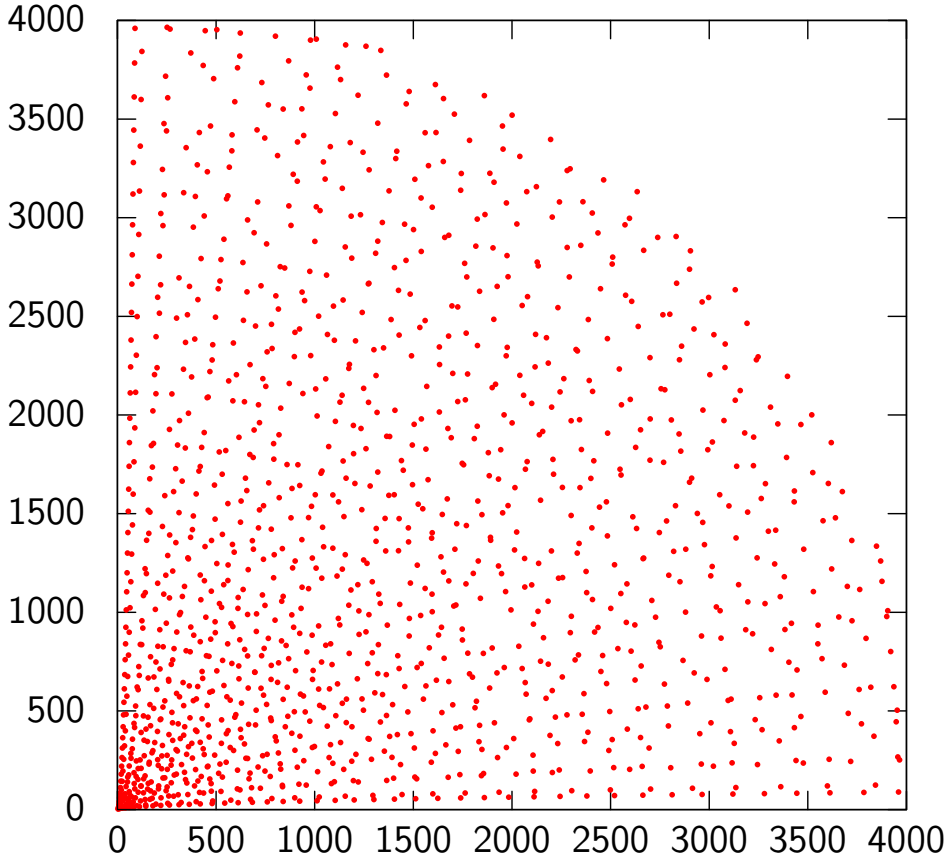
- tables should store **exact values** on machine numbers
- it **saves bits**
- it **concentrates the error** on subsequent steps
- it makes the reconstruction step **easier**

What if we had rational numbers to tabulate?

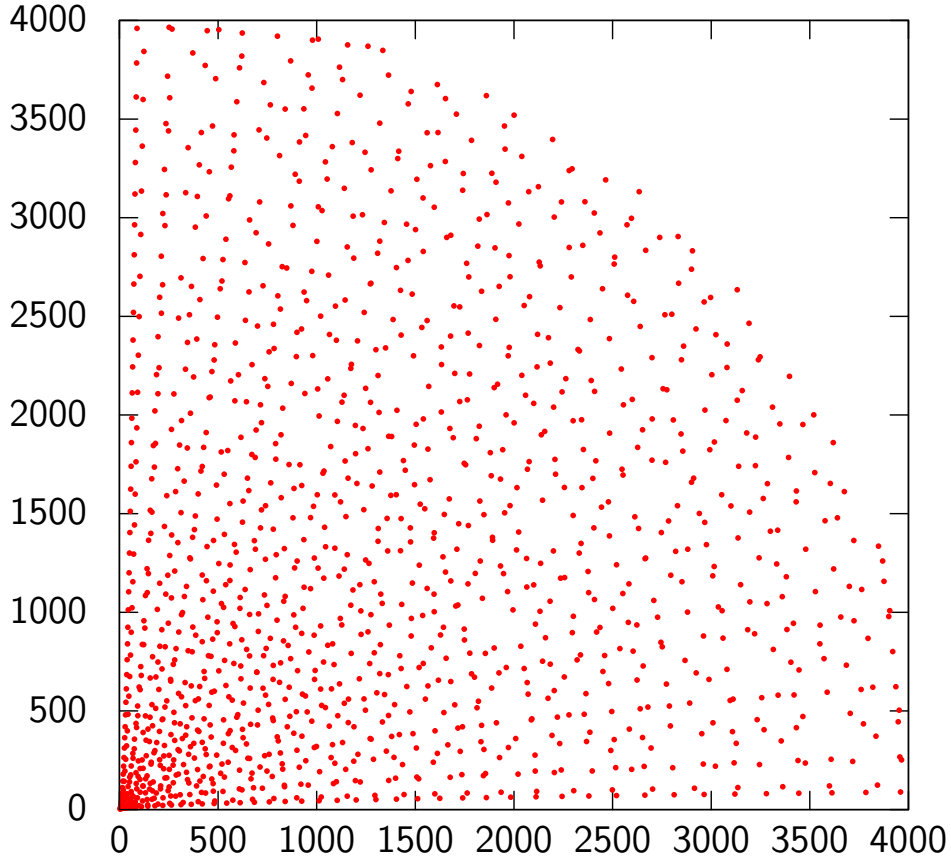


$$\sin(x) \approx (a \otimes \cos(x_\ell) \oplus b \otimes \sin(x_\ell)) \oslash c$$

We've been having them since Euclid ($\approx$ 300 BC)



We've been having them since Euclid ($\approx$ 300 BC)



Each dot represents a primitive Pythagorean triple (PPT):

$$\exists(\mathbf{a},\mathbf{b},\mathbf{c}) \in \mathbb{N}^3 \text{ coprime} \mid a^2 + b^2 = c^2$$

$$\Rightarrow \exists x \in \left[0, \frac{\pi}{2}\right] \mid \sin(\mathbf{x}) = \frac{\mathbf{a}}{\mathbf{c}} \quad \text{and} \quad \cos(\mathbf{x}) = \frac{\mathbf{b}}{\mathbf{c}}$$

Primitive Pythagorean triples (PPT) generation

One easy way: ternary-trees with root $(3, 4, 5)$

- 3 linear relationships $\rightarrow$ 3 children/node, e.g:

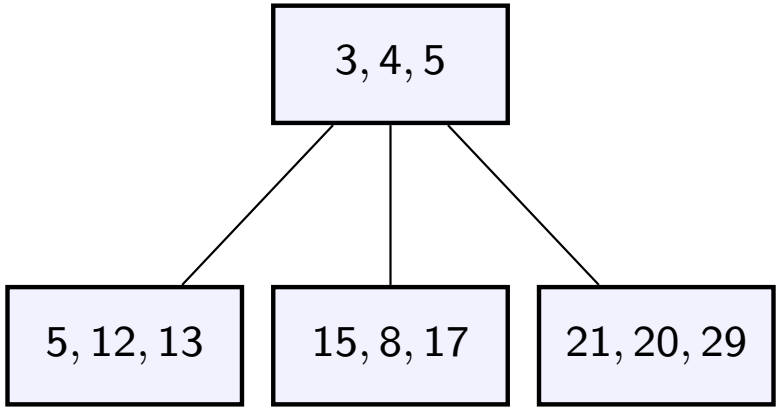
$$\begin{pmatrix} 1 & -2 & 2 \\ 2 & -1 & 2 \\ 2 & -2 & 3 \end{pmatrix}, \begin{pmatrix} -1 & 2 & 2 \\ 2 & -1 & 2 \\ -2 & 2 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 3 \end{pmatrix}$$

Primitive Pythagorean triples (PPT) generation

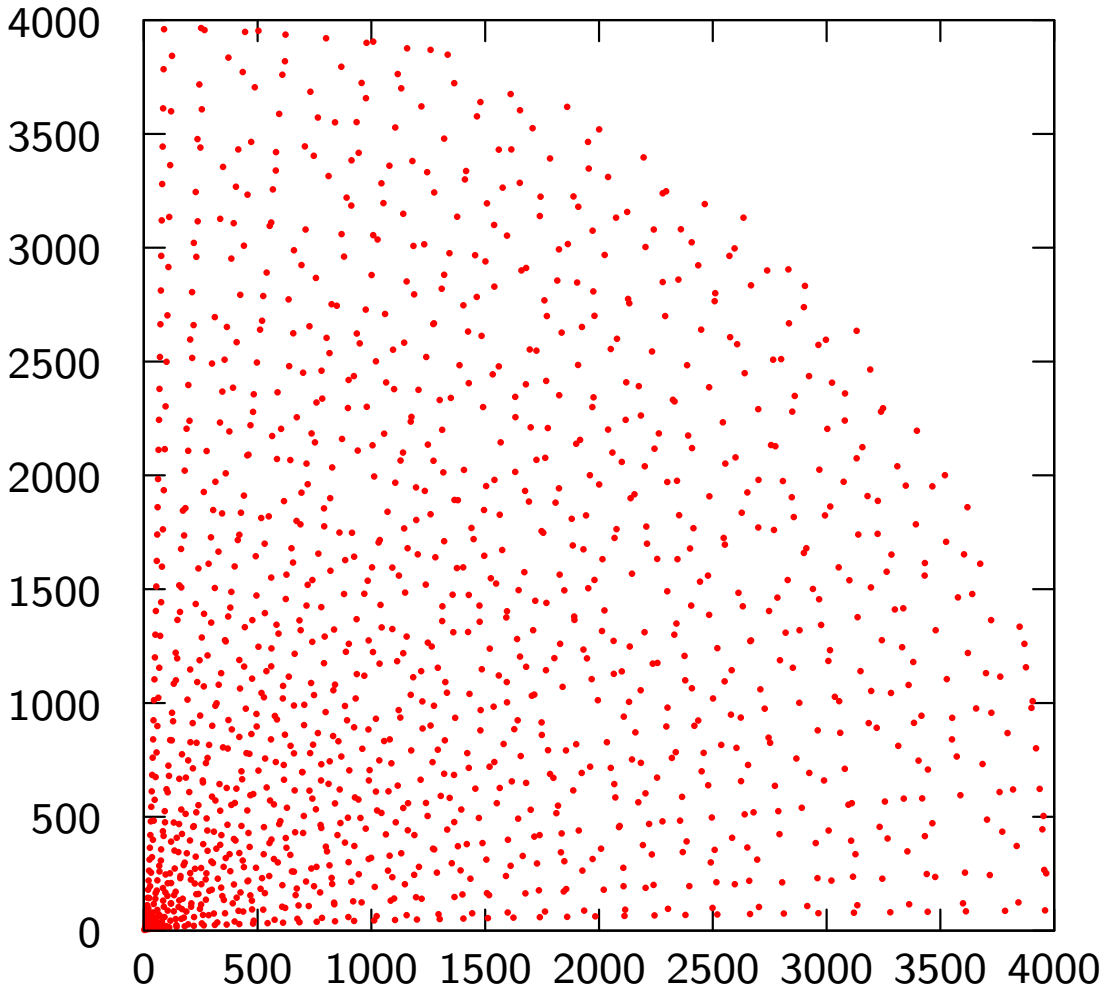
One easy way: ternary-trees with root $(3, 4, 5)$

- 3 linear relationships $\rightarrow$ 3 children/node, e.g:

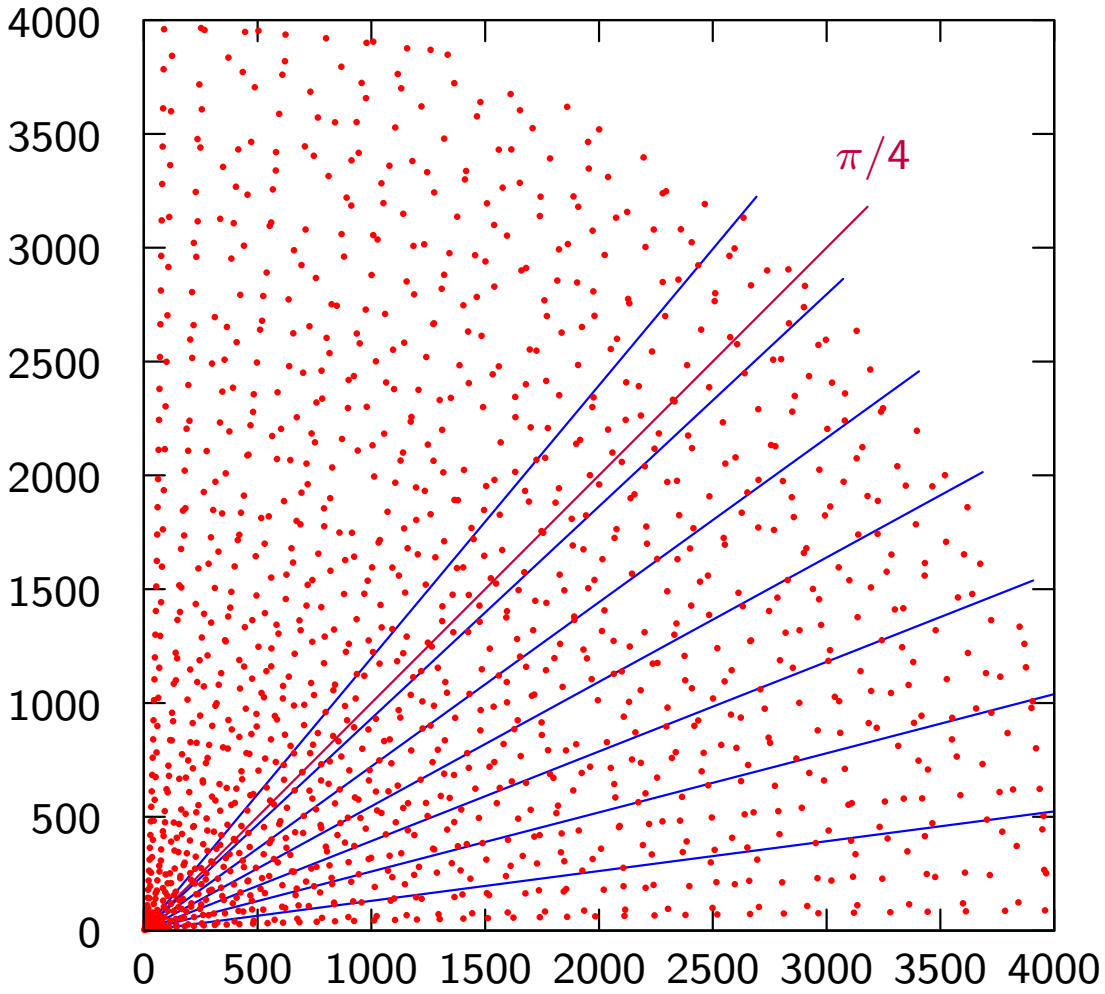
$$\begin{pmatrix} 1 & -2 & 2 \\ 2 & -1 & 2 \\ 2 & -2 & 3 \end{pmatrix}, \begin{pmatrix} -1 & 2 & 2 \\ 2 & -1 & 2 \\ -2 & 2 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 3 \end{pmatrix}$$



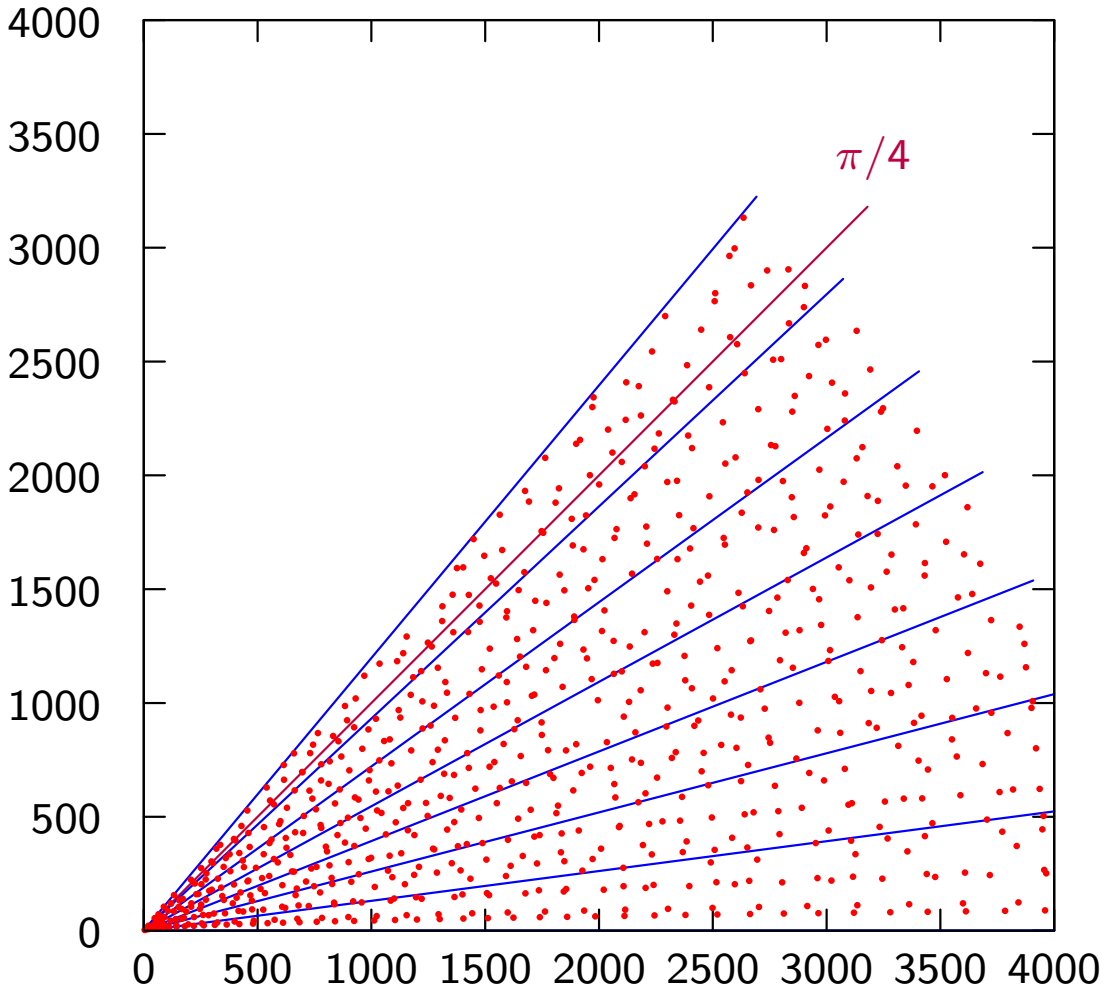
Primitive Pythagorean triples with $c \leq 2^{12}$



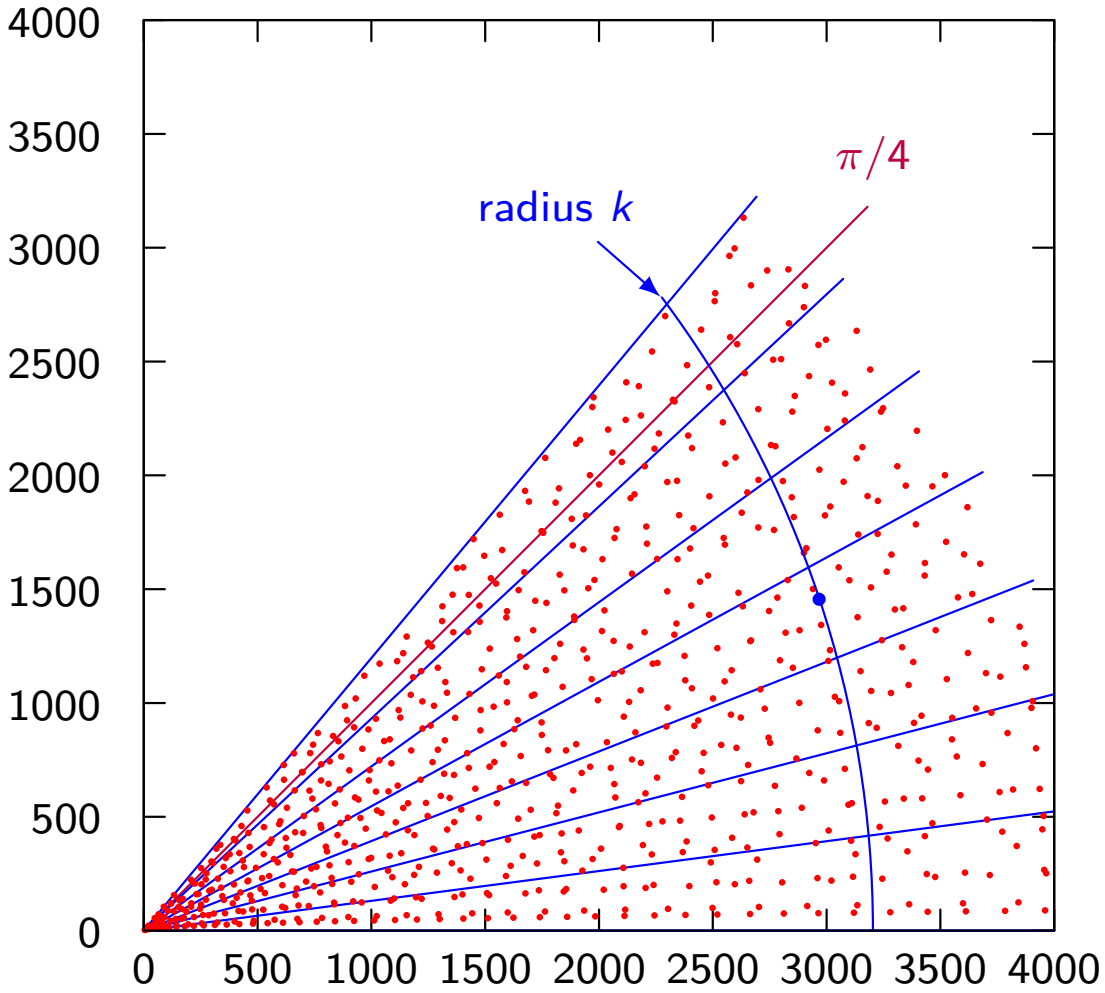
Primitive Pythagorean triples with $c \leq 2^{12}$



Primitive Pythagorean triples with $c \leq 2^{12}$

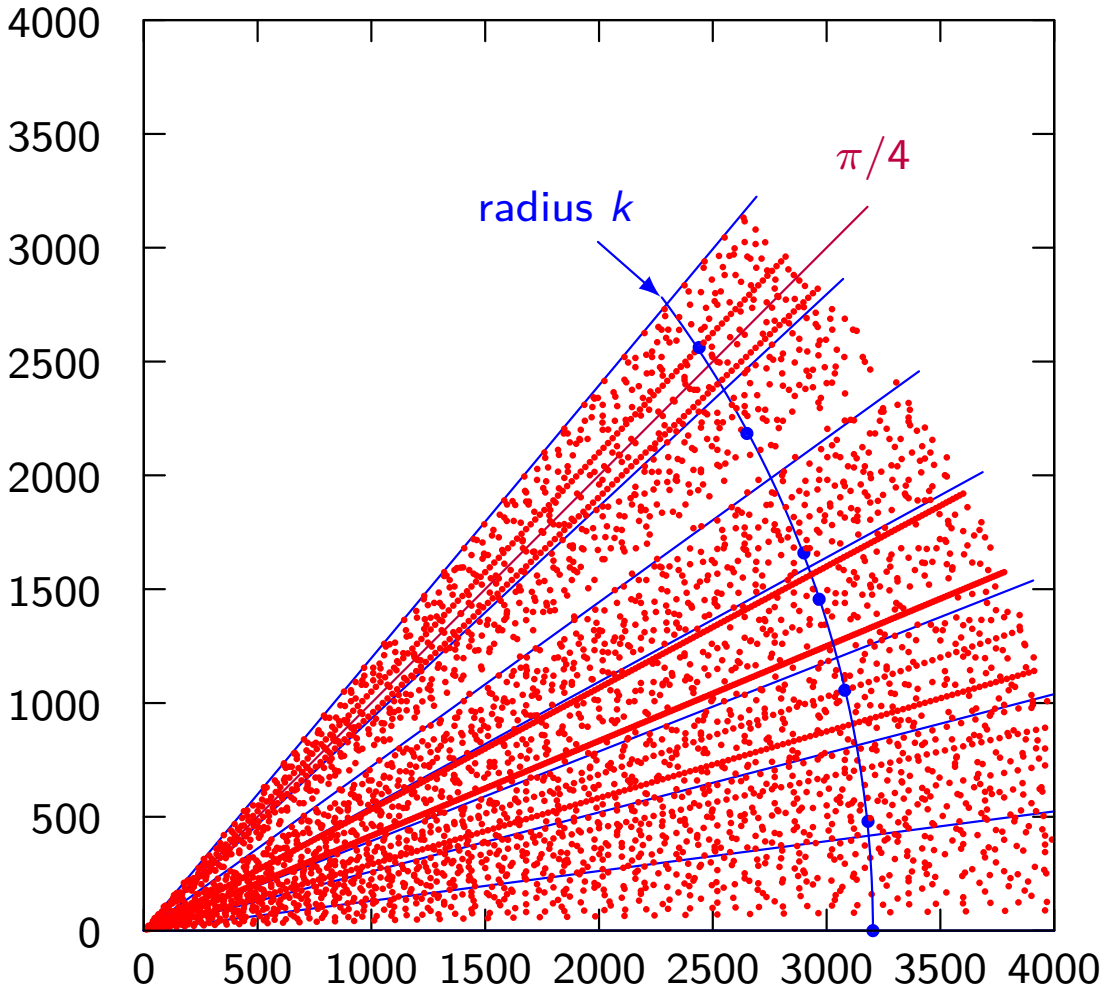


What are "good" Pythagorean triples?



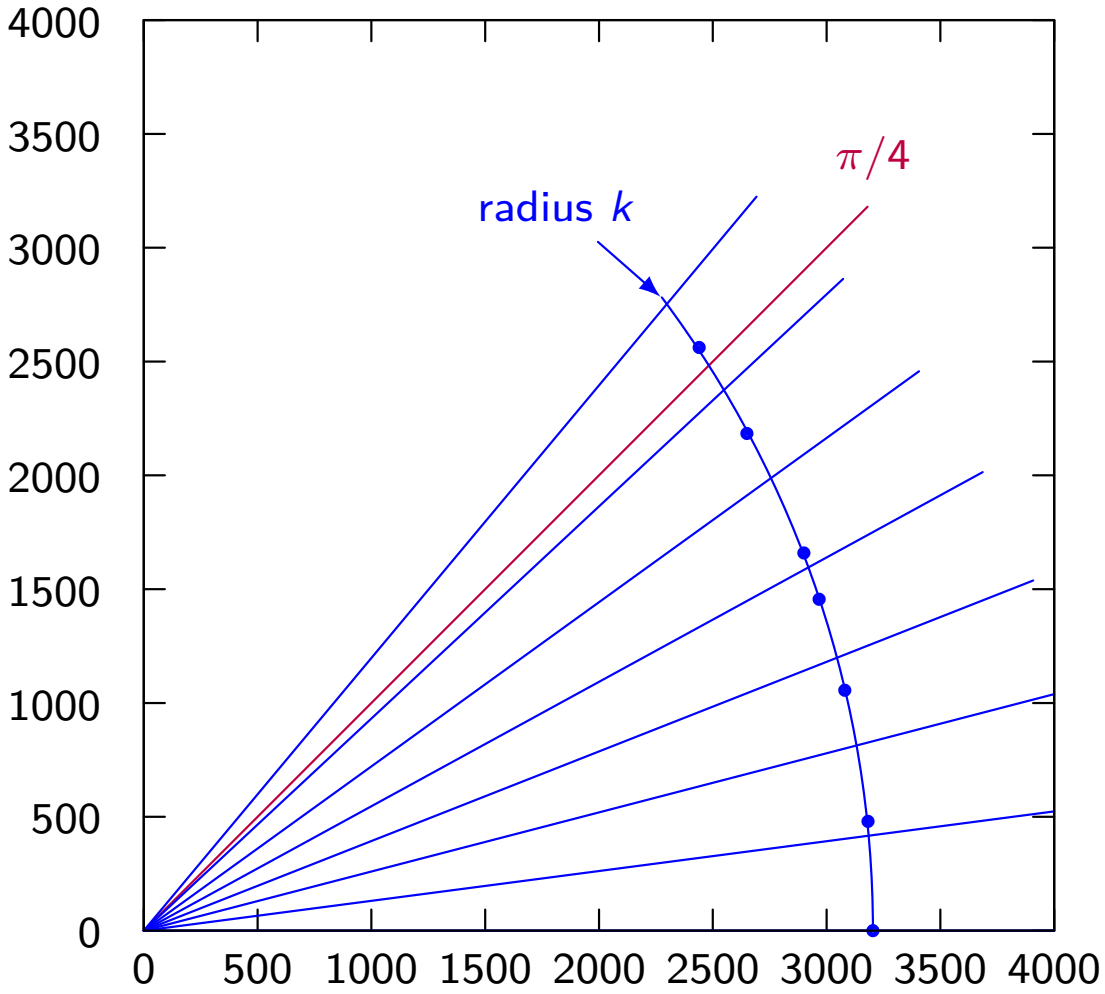
$$\sin(x) \approx (a_i \otimes \cos(x_\ell - corr) \oplus b_i \otimes \sin(x_\ell - corr)) \oslash c_i$$

What are "good" Pythagorean triples?



$$\sin(x) \approx \left(\frac{a_i \cdot k}{c_i} \otimes \cos(x_\ell - corr) \oplus \frac{b_i \cdot k}{c_i} \otimes \sin(x_\ell - corr) \right) \oslash k$$

What are "good" Pythagorean triples?



$$\sin(x) \approx A_i \otimes [\cos(x_\ell - corr)/k] \oplus B_i \otimes [\sin(x_\ell - corr)/k]$$

Actual use of Pythagorean triples

- For each table entry i:
  - store **integers** $A_i = \frac{a_i}{c_i} \cdot k$ and $B_i = \frac{b_i}{c_i} \cdot k$ $(A_i, B_i \in \mathbb{F})$

Actual use of Pythagorean triples

- For each table entry i:
  - store **integers** $A_i = \frac{a_i}{c_i} \cdot k$ and $B_i = \frac{b_i}{c_i} \cdot k$ $(A_i, B_i \in \mathbb{F})$
- Incorporate $\frac{1}{k}$ into polynomial approximants:

$$\frac{\sin(x_\ell - corr)}{k} \quad \text{and} \quad \frac{\cos(x_\ell - corr)}{k}$$

- k should be ***a small*** **least common multiple** (LCM) of **any combination** of one hypotenuse c per table entry
  - small $\rightarrow$ each A_i and B_i fits in a machine word

Table generation algorithm

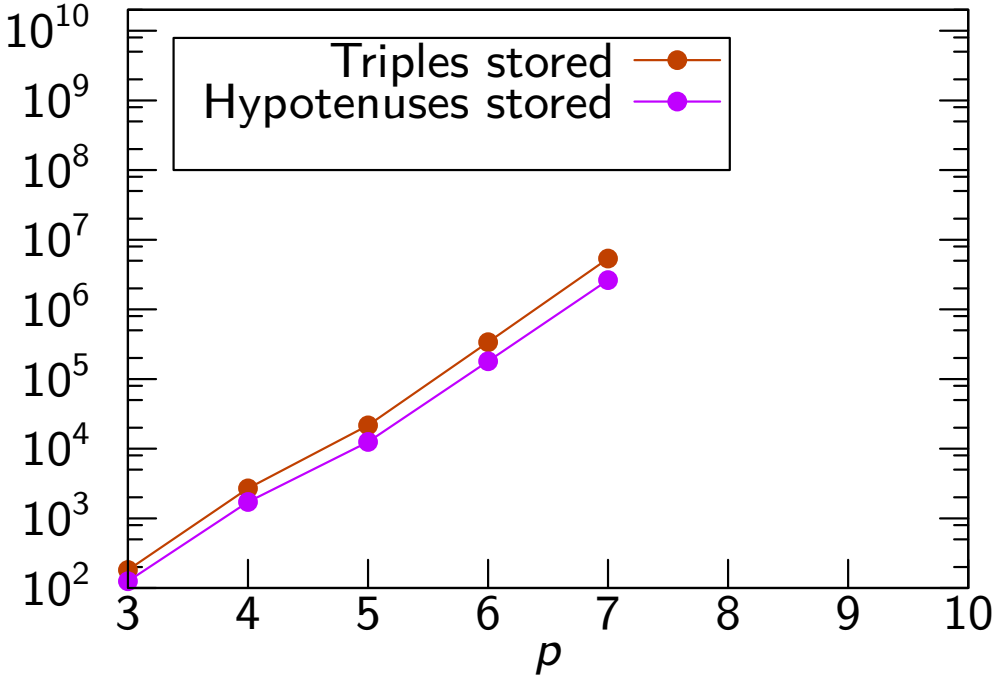
- **Input**
  - p: table index size
- **Algorithm steps**

```
n ← 4
repeat
    Generate all PPTs (a, b, c) such that c ≤ 2^n.
    Search for the LCM k among all generated hypotenuses c.
    n ← n + 1
until such a k is found
Build tabulated values (A, B, corr) for every entry.
```

Exhaustive search

Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60 GHz (32 cores) 125 GB RAM

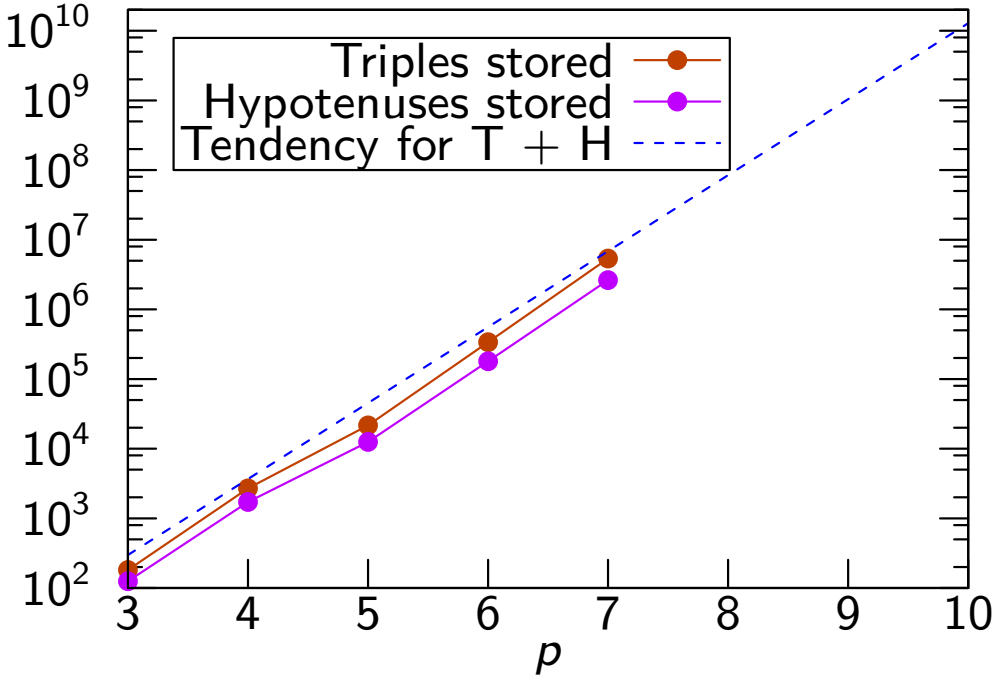
p	k_{min}	n	Time (s)
3	725	10	$\ll 1$
4	10,625	14	0.01
5	130,645	17	0.14
6	1,676,285	21	6
7	32,846,125	25	1000



Exhaustive search

Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60 GHz (32 cores) 125 GB RAM

p	k_{min}	n	Time (s)
3	725	10	$\ll 1$
4	10,625	14	0.01
5	130,645	17	0.14
6	1,676,285	21	6
7	32,846,125	25	1000



- Bottlenecks = **searching** the LCM and **memory**
- Finding k for a 10-bit addressed table is **desperate** with this exhaustive technique
  - our estimations show that $n = \lceil \log_2(k_{min}) \rceil \approx 37$

Heuristic search

p	k_{min}	Prime Factorization	Triples
5	130,645	$5 \cdot 17 \cdot 29 \cdot 53$	21,588
6	1,676,285	$5 \cdot 13 \cdot 17 \cdot 37 \cdot 41$	338,660
7	32,846,125	$5^3 \cdot 13 \cdot 17 \cdot 29 \cdot 41$	5,365,290

Observation: k is always a product of ***small* Pythagorean primes**

- Pythagorean primes less than 70:
  `p = {5, 13, 17, 29, 37, 41, 53, 61}`

Heuristic search

p	k_{min}	Prime Factorization	Triples
5	130,645	$5 \cdot 17 \cdot 29 \cdot 53$	21,588
6	1,676,285	$5 \cdot 13 \cdot 17 \cdot 37 \cdot 41$	338,660
7	32,846,125	$5^3 \cdot 13 \cdot 17 \cdot 29 \cdot 41$	5,365,290

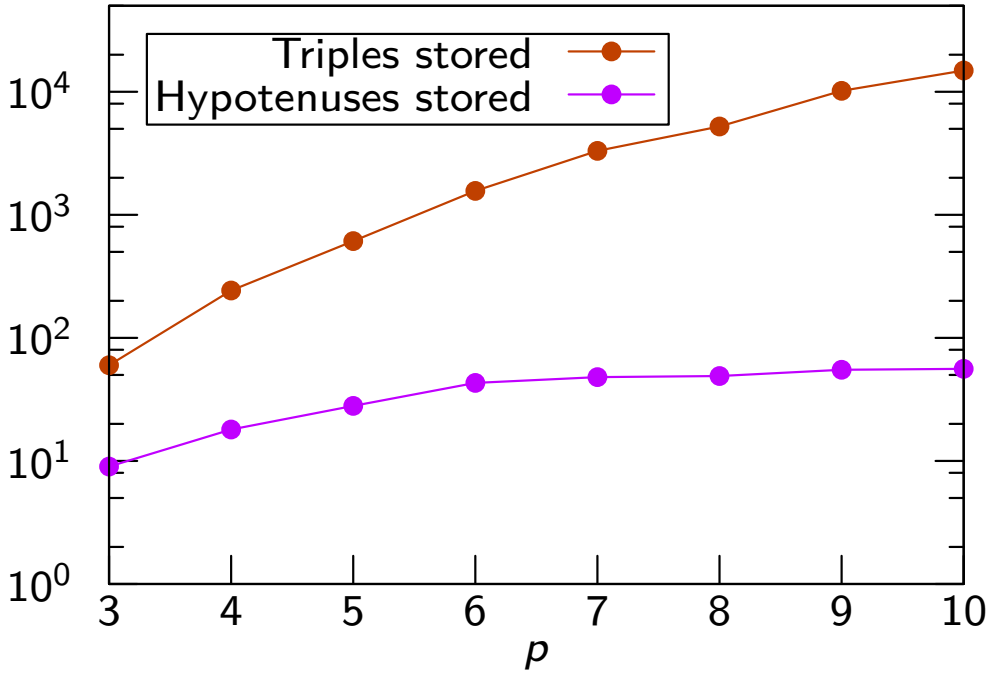
Observation: k is always a product of ***small* Pythagorean primes**

- Pythagorean primes less than 70:
  `p = {5, 13, 17, 29, 37, 41, 53, 61}`
- During the generation, **only keep triples with hypotenuse**

$$c = \prod_i p[i]^{r_i} \quad \text{with} \quad \begin{cases} \boldsymbol{r_i = 0 \text{ or } 1} & \textbf{if } \boldsymbol{p[i] \neq 5} \\ \boldsymbol{r_i \in \mathbb{N}} & \textbf{if } \boldsymbol{p[i] = 5} \end{cases}$$

Heuristic search results

p	k_{min}	n	Time (s)
6	1,676,285	21	0.12
7	32,846,125	25	0.88
8	243,061,325	28	4.63
9	12,882,250,225	34	269
10	370,668,520,625	39	8563



- **Same results** as the exhaustive search for $p \leq 7$
- **Exponentially better** than exhaustive search with respect to p
- New **bottleneck**: heuristic test over $\approx 2^n$ hypotenuses

Comparisons with other table-based methods

- Correct rounding in double precision
  - 2-step Ziv strategy:
    - quick phase accurate to 2^{-66}
    - slow phase accurate to 2^{-150}
- Table index size $p = 10$ bits
- Estimate memory accesses (MA), FLOPs, and table size

Method	Table size (B)	Quick phase	Slow phase
Tang	38,640	4 MA + 64 FLOP	6 MA + 241 FLOP
Gal	57,960	**3 MA + 53 FLOP**	9 MA + 268 FLOP
Proposed	32,200	**3 MA + 53 FLOP**	**5 MA + 148 FLOP**

- Table-size 17% lower than Tang's, 45% lower than Gal's
- Quick phase 25% + 17% less expensive than Tang's, same as Gal's
- Slow phase 17% + 39% less expensive than Tang's, 45% + 45% than Gal's

Comparisons with other table-based methods

- Correct rounding in double precision
  - 2-step Ziv strategy:
    - quick phase accurate to 2^{-66}
    - slow phase accurate to 2^{-150}
- Table index size $p = 10$ bits
- Estimate memory accesses (MA), FLOPs, and table size

Method	Table size (B)	Quick phase	Slow phase
Tang	38,640	4 MA + 64 FLOP	6 MA + 241 FLOP
Gal	57,960	**3 MA + 53 FLOP**	9 MA + 268 FLOP
Proposed	32,200	**3 MA + 53 FLOP**	**5 MA + 148 FLOP**

- Table-size 17% lower than Tang's, 45% lower than Gal's
- Quick phase 25% + 17% less expensive than Tang's, same as Gal's
- Slow phase 17% + 39% less expensive than Tang's, 45% + 45% than Gal's

Comparisons with other table-based methods

- Correct rounding in double precision
  - 2-step Ziv strategy:
    - quick phase accurate to 2^{-66}
    - slow phase accurate to 2^{-150}
- Table index size $p = 10$ bits
- Estimate memory accesses (MA), FLOPs, and table size

Method	Table size (B)	Quick phase	Slow phase
Tang	38,640	4 MA + 64 FLOP	6 MA + 241 FLOP
Gal	57,960	**3 MA + 53 FLOP**	9 MA + 268 FLOP
Proposed	32,200	**3 MA + 53 FLOP**	**5 MA + 148 FLOP**

- Table-size 17% lower than Tang's, 45% lower than Gal's
- Quick phase 25% + 17% less expensive than Tang's, same as Gal's
- Slow phase 17% + 39% less expensive than Tang's, 45% + 45% than Gal's

Comparisons with other table-based methods

- Correct rounding in double precision
  - 2-step Ziv strategy:
    - quick phase accurate to 2^{-66}
    - slow phase accurate to 2^{-150}
- Table index size $p = 10$ bits
- Estimate memory accesses (MA), FLOPs, and table size

Method	Table size (B)	Quick phase	Slow phase
Tang	38,640	4 MA + 64 FLOP	6 MA + 241 FLOP
Gal	57,960	**3 MA + 53 FLOP**	9 MA + 268 FLOP
Proposed	32,200	**3 MA + 53 FLOP**	**5 MA + 148 FLOP**

- Table-size 17% lower than Tang's, 45% lower than Gal's
- Quick phase 25% + 17% less expensive than Tang's, same as Gal's
- Slow phase 17% + 39% less expensive than Tang's, 45% + 45% than Gal's

Conclusion and Perspectives

We propose:

- a **new algorithm** for table-based range reductions:
  - using **error-free** values (integers)
  - **concentrating** the error on the other steps
  - estimated gains of **45% in table-size, FLOPs and memory accesses** for correctly-rounded double precision implementations
- a **prototype** able to pre-compute tables up to 10 indexing bits

Perspectives:

- Evaluation of **accuracy and performance** in available libraries
- Consider **other functions** $(\sinh, \cosh, \ldots)$
- **Integration** in a full code-generation chain (MetaLibm)

Range Reduction Based on Pythagorean Triples for Trigonometric Function Evaluation

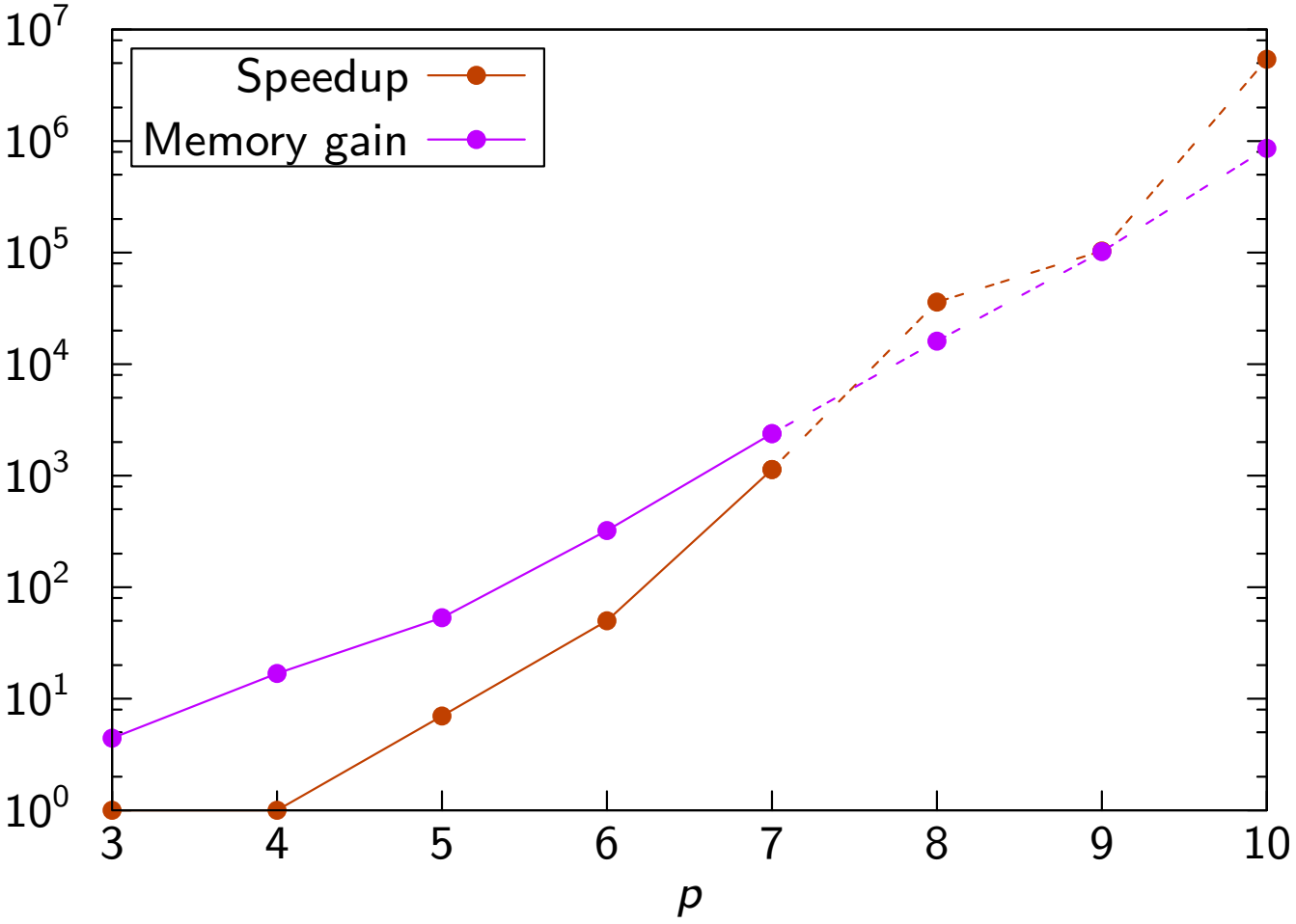
IEEE ASAP 2015 Conference

Hugues de Lassus Saint-Geniès David Defour Guillaume Revy

DALI, Université de Perpignan Via Domitia
LIRMM, Université de Montpellier
CNRS UMR 5506

July 27th, 2015

Heuristic gains



- **Exponential speedup**
- **Exponential memory reduction**