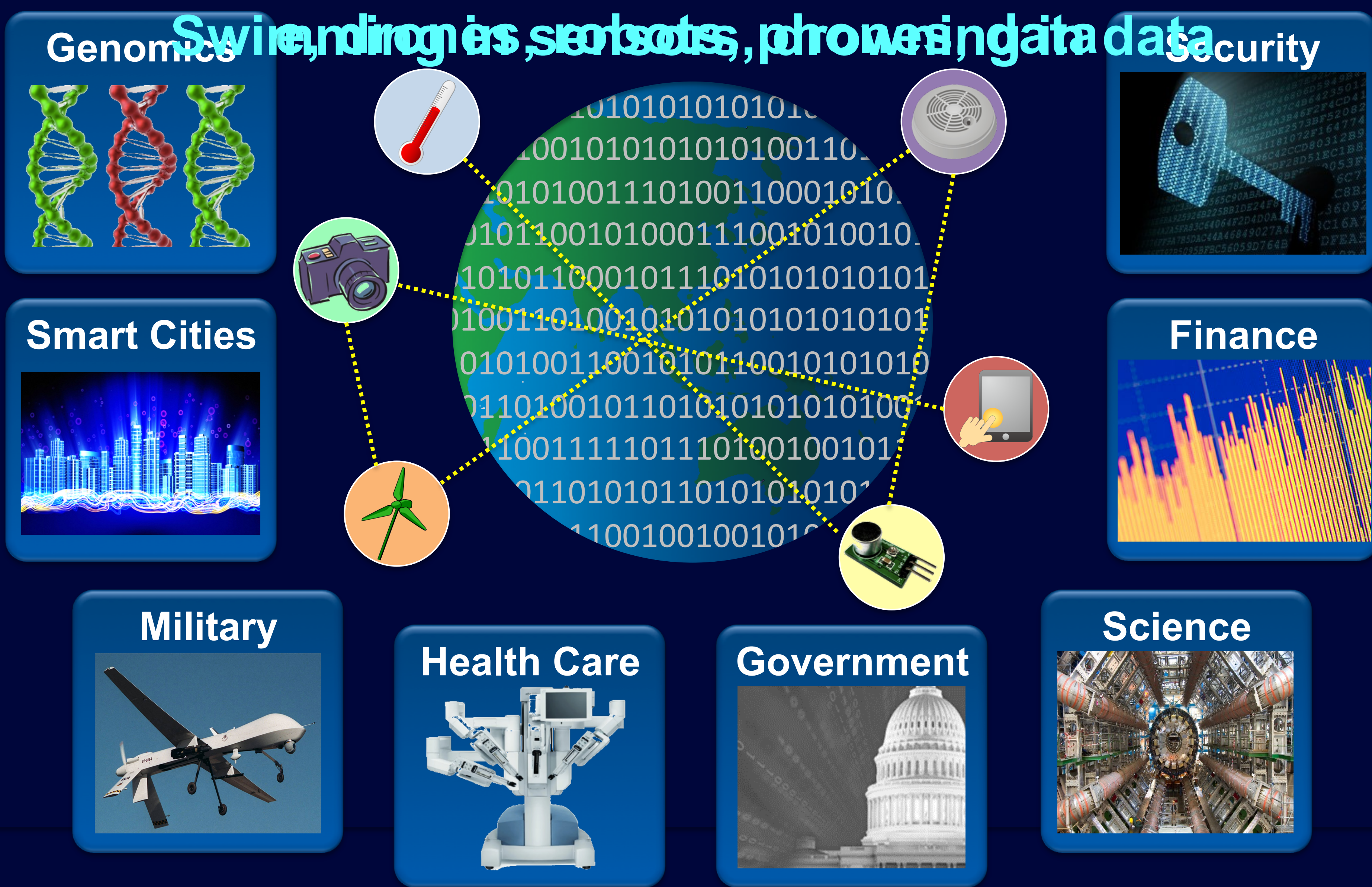
Robust Computing Systems

From Today to the N3XT 1,000×

Subhasish Mitra

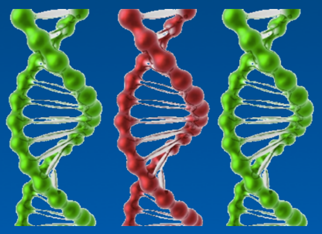
Department of EE & Department of CS
Stanford University

World Relies on Computing



World Relies on Computing

Genomics

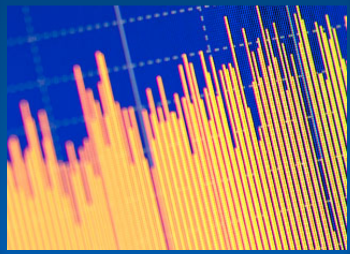


Smart Cities

- Robust operation
- Performance
- New application horizon

Security

Finance

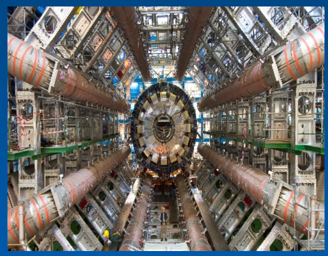


Military

Health Care

Government

Science



Research Topics

- Robust operation
  - Bugs, reliability, security
- Revolutionize NanoSystems
  - *1,000*× opportunity
- Program human brain
  - Stanford Big Ideas in Neuroscience

Outline

- Design bugs: QED & Symbolic QED

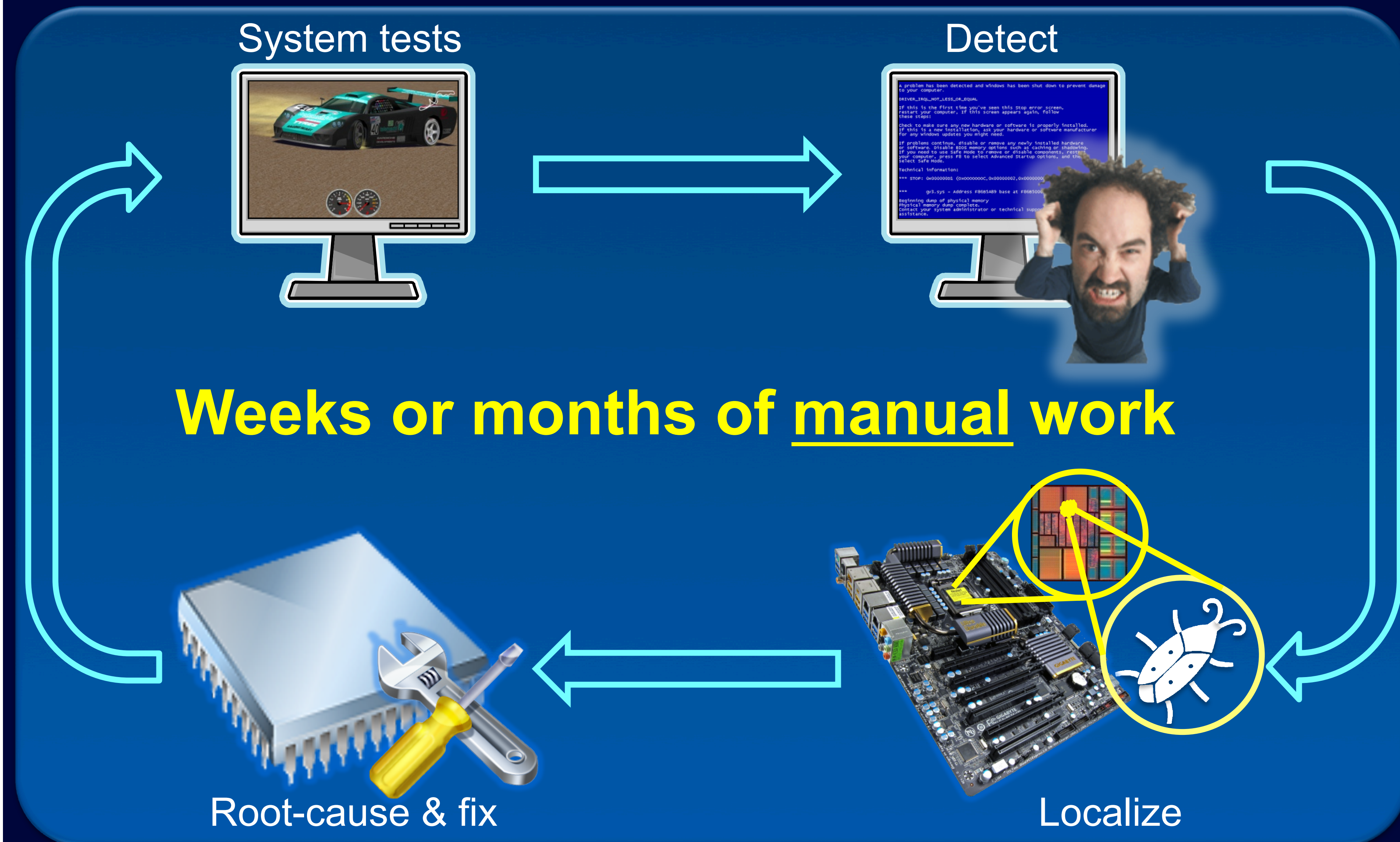
Pre-Silicon Verification Inadequate

Design → Pre-silicon Verification → Fab → **Post-silicon Validation** → High Volume

It's only getting worse: custom hardware

Source: Intel

Post-Silicon Validation Difficult



Scalability Barriers

- System-level failure reproduction
- Full system simulation

J. Stinson (ex-Intel)

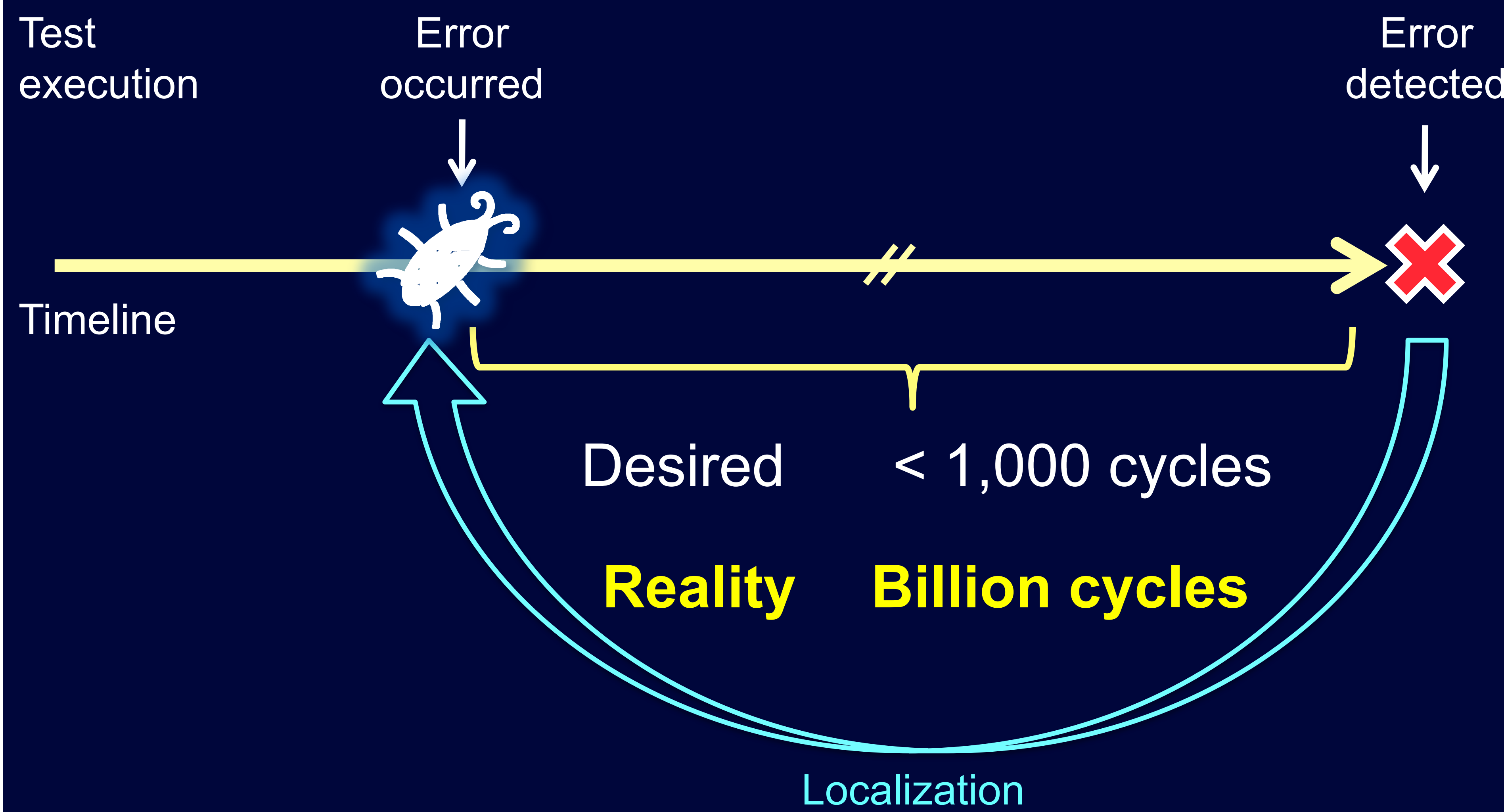
Post-silicon costs rising faster than design cost

QED

- Post-silicon
  - Electrical bugs, logic bugs

[Lin IEEE TCAD 14, Singh CAV 17]

Error Detection Latency

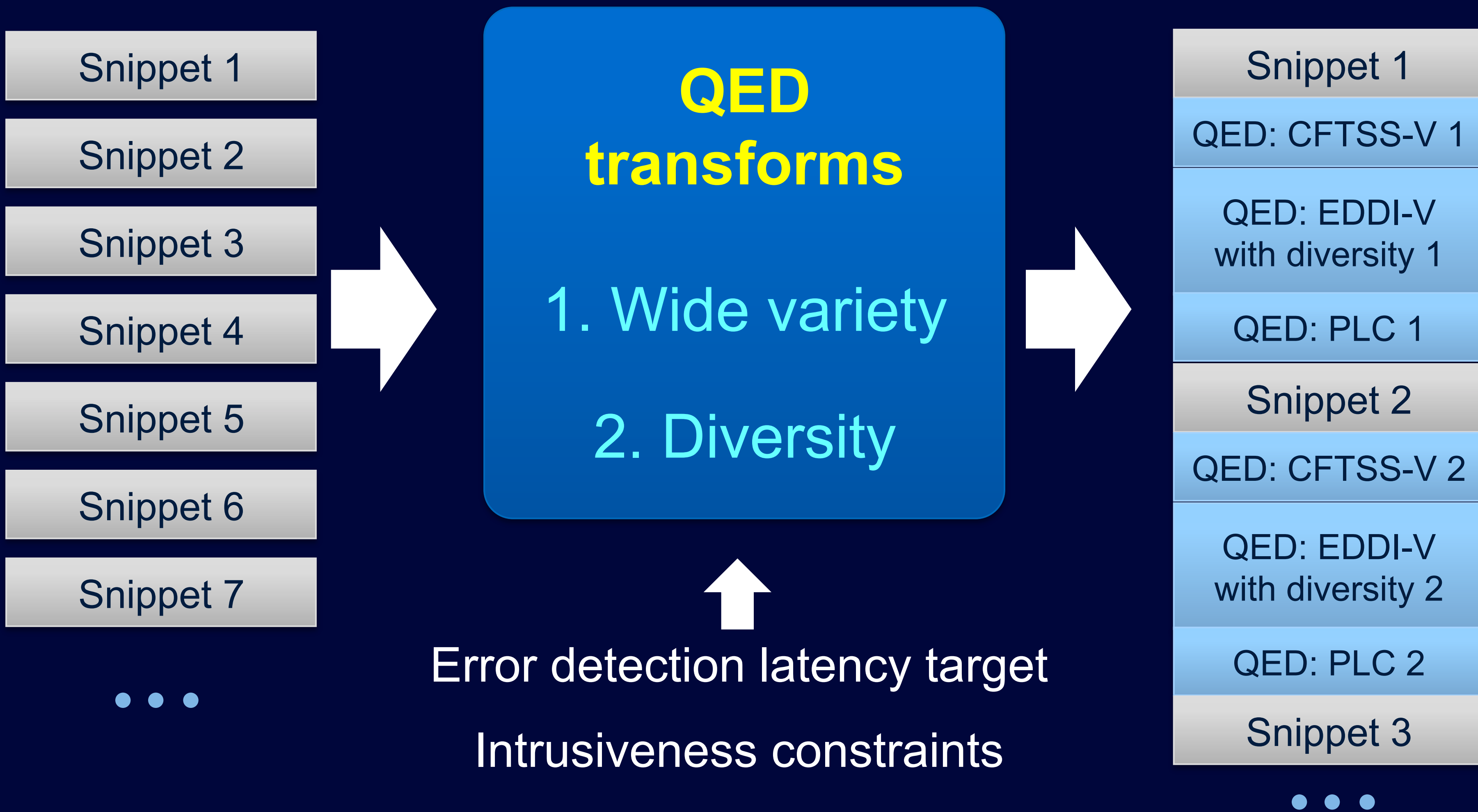


Quick Error Detection

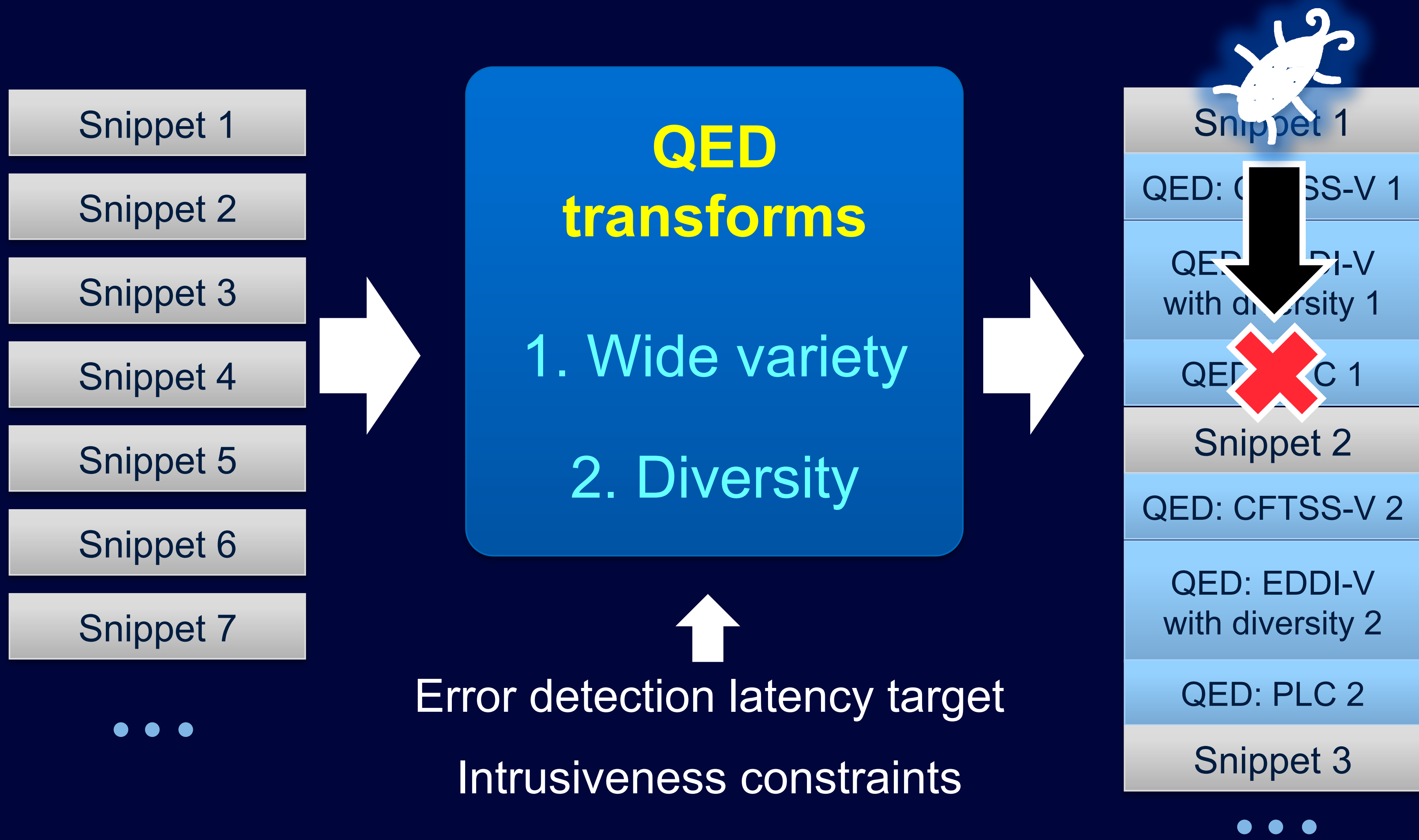
- Error detection latency: guaranteed short
- Coverage: improved
- Software-only: readily applicable

[Lin IEEE TCAD 14]

Quick Error Detection

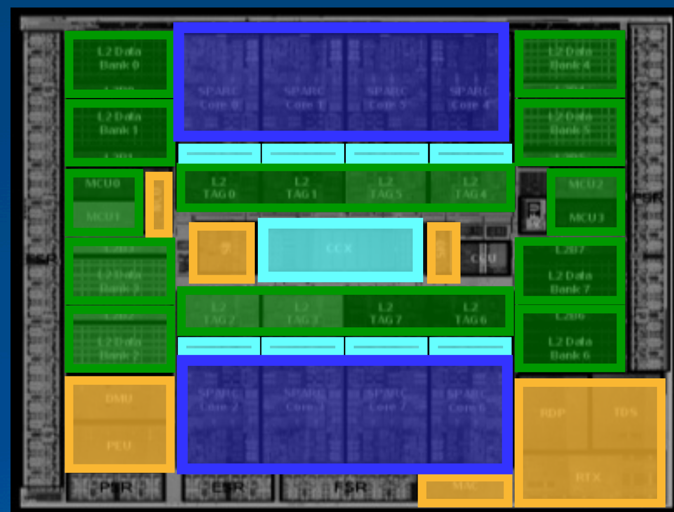


Quick Error Detection



QED Transforms

System on Chip



Processor cores

Uncore, accelerators

EDDI-V

CFTSS-V

CFCSS-V

PLC

Fast QED

Hybrid QED

QED Example: Duplicate & Check

Validation program

```
void matgen (REAL a[], int lda, int n, REAL b[], REAL *norma)
{
    int init, i, j;
    init = 1325;
    *norma = 0.0;
    for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
            init = 3125*init % 0x100000000l;
            a[lda*j+i] = (init - 32768.0)/16384.0;
            *norma = (a[lda*j+i] > *norma) ? a[lda*j+i] : *norma;
        }
    }
    for (i = 0; i < n; i++) {
        b[i] = 0.0;
    }
    for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
            b[i] = b[i] + a[lda*j+i];
        }
    }
    return;
}

void dgefa(REAL a[], int lda, int n, int ipvt[], int *info){
    /*     internal variables       */
    REAL t;
    int j,k,kp1,l,nm1;

    /*     gaussian elimination with partial pivoting       */

    *info = 0;
    nm1 = n - 1;
    if (nm1 >=  0) {
        for (k = 0; k < nm1; k++) {
            kp1 = k + 1;

            /* find l = pivot index */
            l = idamax(n-k,&a[lda*k+k],1) + k;
            ipvt[k] = l;

            /* zero pivot implies this column already
               triangularized */

            if (a[lda*k+l] != ZERO) {
                /* interchange if necessary */

                if (l != k) {
                    t = a[lda*k+l];
                    a[lda*k+l] = a[lda*k+k];
                    a[lda*k+k] = t;
                }

                /* compute multipliers */
                t = -ONE/a[lda*k+k];
                dscal(n-(k+1),t,&a[lda*k+k+1],1);

                /* row elimination with column indexing */
                for (j = kp1; j < n; j++) {
                    t = a[lda*j+l];
                    if (l != k) {
                        a[lda*j+l] = a[lda*j+k];
                        a[lda*j+k] = t;
                    }
                    daxpy(n-(k+1),t,&a[lda*k+k+1],1,
                    &a[lda*j+k+1],1);
                }
            }
            else {
                *info = k;
            }
        }
    }

    ipvt[n-1] = n-1;
    if (a[lda*(n-1)+(n-1)] == ZERO) *info = n-1;
    checkExpected(a, ipvt, info);
    return;
}

void dgesl(REAL a[],int lda,int n,int ipvt[],REAL b[],int job )
{
    REAL t;
    int k,kb,l,nm1;

    nm1 = n - 1;
    if (job == 0) {
        /* job = 0 , solve  a * x = b
           first solve  l*y = b          */
        if (nm1 >= 1) {
            for (k = 0; k < nm1; k++) {
                l = ipvt[k];
                t = b[l];
                if (l != k){
                    b[l] = b[k];
                    b[k] = t;
                }
                daxpy(n-(k+1),t,&a[lda*k+k+1],1,&b[k+1],1 );
            }
        }

        /* now solve  u*x = y */
        for (kb = 0; kb < n; kb++) {
            k = n - (kb + 1);
            b[k] = b[k]/a[lda*k+k];
            t = -b[k];
            daxpy(k,t,&a[lda*k+0],1,&b[0],1 );
        }
    }
    else {
```

Trace

• • •

```
R1  ← R1  + 5
R2  ← R2  - R1
R3  ← R1  * R2
```

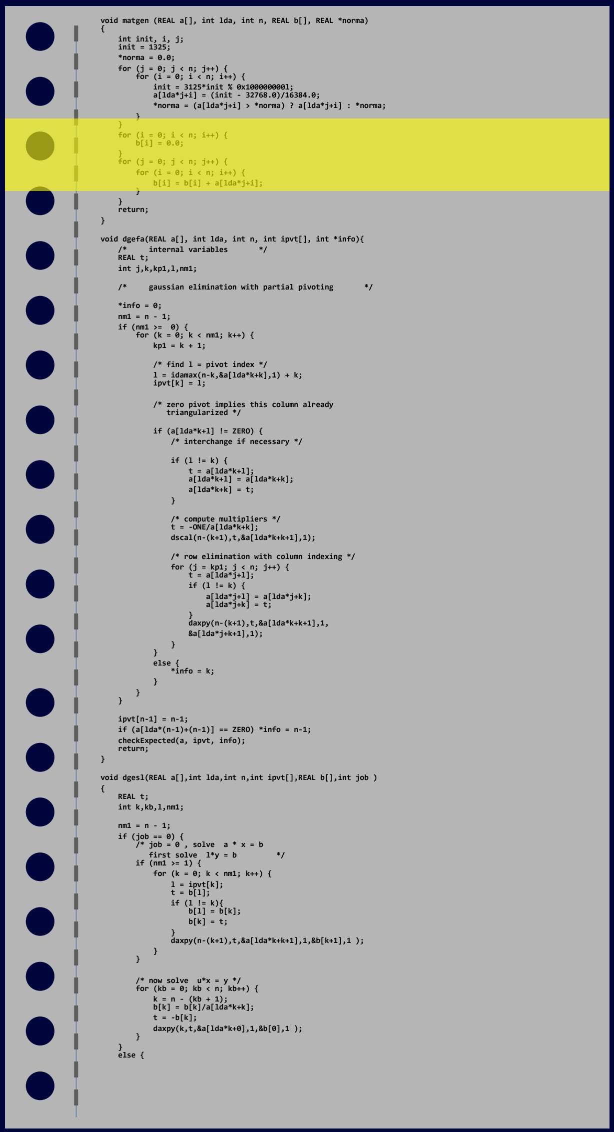
Very Long!

• • •

`Check end_result`

QED Example: Duplicate & Check

Validation program



```
void matgen (REAL a[], int lda, int n, REAL b[], REAL *norma)
{
    int init, i, j;
    init = 1325;
    *norma = 0.0;
    for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
            init = 3125*init % 0x100000000l;
            a[lda*j+i] = (init - 32768.0)/16384.0;
            *norma = (a[lda*j+i] > *norma) ? a[lda*j+i] : *norma;
        }
    }
    for (i = 0; i < n; i++) {
        b[i] = 0.0;
    }
    for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
            b[i] = b[i] + a[lda*j+i];
        }
    }
    return;
}

void dgefa(REAL a[], int lda, int n, int ipvt[], int *info){
    /*     internal variables       */
    REAL t;
    int j,k,kp1,l,nm1;

    /*     gaussian elimination with partial pivoting       */

    *info = 0;
    nm1 = n - 1;
    if (nm1 >=  0) {
        for (k = 0; k < nm1; k++) {
            kp1 = k + 1;

            /* find l = pivot index */
            l = idamax(n-k,&a[lda*k+k],1) + k;
            ipvt[k] = l;

            /* zero pivot implies this column already
               triangularized */

            if (a[lda*k+l] != ZERO) {
                /* interchange if necessary */

                if (l != k) {
                    t = a[lda*k+l];
                    a[lda*k+l] = a[lda*k+k];
                    a[lda*k+k] = t;
                }

                /* compute multipliers */
                t = -ONE/a[lda*k+k];
                dscal(n-(k+1),t,&a[lda*k+k+1],1);

                /* row elimination with column indexing */
                for (j = kp1; j < n; j++) {
                    t = a[lda*j+l];
                    if (l != k) {
                        a[lda*j+l] = a[lda*j+k];
                        a[lda*j+k] = t;
                    }
                    daxpy(n-(k+1),t,&a[lda*k+k+1],1,
                    &a[lda*j+k+1],1);
                }
            }
            else {
                *info = k;
            }
        }
    }

    ipvt[n-1] = n-1;
    if (a[lda*(n-1)+(n-1)] == ZERO) *info = n-1;
    checkExpected(a, ipvt, info);
    return;
}

void dgesl(REAL a[],int lda,int n,int ipvt[],REAL b[],int job )
{
    REAL t;
    int k,kb,l,nm1;

    nm1 = n - 1;
    if (job == 0) {
        /* job = 0 , solve  a * x = b
           first solve  l*y = b         */
        if (nm1 >= 1) {
            for (k = 0; k < nm1; k++) {
                l = ipvt[k];
                t = b[l];
                if (l != k){
                    b[l] = b[k];
                    b[k] = t;
                }
                daxpy(n-(k+1),t,&a[lda*k+k+1],1,&b[k+1],1 );
            }
        }

        /* now solve  u*x = y */
        for (kb = 0; kb < n; kb++) {
            k = n - (kb + 1);
            b[k] = b[k]/a[lda*k+k];
            t = -b[k];
            daxpy(k,t,&a[lda*k+0],1,&b[0],1 );
        }
    }
    else {
```

QED Trace

```
R16 ← R1; R18 ← R2; R19 ← R3

    • • •

R1  ← R1  + 5
R16 ← R16 + 5
R1  == R16
R2  ← R2  – R1
R18 ← R18 – R16
R2  == R18
R3  ← R1  * R2
R19 ← R16 * R18
R3  == R19

    • • •

Check end_result
```

Quick!

QED Improves Coverage

Validation program

```
void matgen (REAL a[], int lda, int n, REAL b[], REAL *norma)
{
    int init, i, j;
    init = 1325;
    *norma = 0.0;
    for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
            init = 3125*init % 0x100000000l;
            a[lda*j+i] = (init - 32768.0)/16384.0;
            *norma = (a[lda*j+i] > *norma) ? a[lda*j+i] : *norma;
        }
    }
    for (i = 0; i < n; i++) {
        b[i] = 0.0;
    }
    for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
            b[i] = b[i] + a[lda*j+i];
        }
    }
    return;
}

void dgefa(REAL a[], int lda, int n, int ipvt[], int *info){
    /*     internal variables       */
    REAL t;
    int j,k,kp1,l,nm1;

    /*     gaussian elimination with partial pivoting       */

    *info = 0;
    nm1 = n - 1;
    if (nm1 >=  0) {
        for (k = 0; k < nm1; k++) {
            kp1 = k + 1;

            /* find l = pivot index */
            l = idamax(n-k,&a[lda*k+k],1) + k;
            ipvt[k] = l;

            /* zero pivot implies this column already
               triangularized */

            if (a[lda*k+l] != ZERO) {
                /* interchange if necessary */

                if (l != k) {
                    t = a[lda*k+l];
                    a[lda*k+l] = a[lda*k+k];
                    a[lda*k+k] = t;
                }

                /* compute multipliers */
                t = -ONE/a[lda*k+k];
                dscal(n-(k+1),t,&a[lda*k+k+1],1);

                /* row elimination with column indexing */
                for (j = kp1; j < n; j++) {
                    t = a[lda*j+l];
                    if (l != k) {
                        a[lda*j+l] = a[lda*j+k];
                        a[lda*j+k] = t;
                    }
                    daxpy(n-(k+1),t,&a[lda*k+k+1],1,
                    &a[lda*j+k+1],1);
                }
            }
            else {
                *info = k;
            }
        }
    }

    ipvt[n-1] = n-1;
    if (a[lda*(n-1)+(n-1)] == ZERO) *info = n-1;
    checkExpected(a, ipvt, info);
    return;
}

void dgesl(REAL a[],int lda,int n,int ipvt[],REAL b[],int job )
{
    REAL t;
    int k,kb,l,nm1;

    nm1 = n - 1;
    if (job == 0) {
        /* job = 0 , solve  a * x = b
           first solve  l*y = b         */
        if (nm1 >= 1) {
            for (k = 0; k < nm1; k++) {
                l = ipvt[k];
                t = b[l];
                if (l != k){
                    b[l] = b[k];
                    b[k] = t;
                }
                daxpy(n-(k+1),t,&a[lda*k+k+1],1,&b[k+1],1 );
            }
        }

        /* now solve  u*x = y */
        for (kb = 0; kb < n; kb++) {
            k = n - (kb + 1);
            b[k] = b[k]/a[lda*k+k];
            t = -b[k];
            daxpy(k,t,&a[lda*k+0],1,&b[0],1 );
        }
    }
    else {
```

Trace

• • •

R1 ← R1 + 1

R1

3 ~~2~~

• • •

R1 ← 0

• • •

Check R1 == 0

Error masked!

QED Improves Coverage

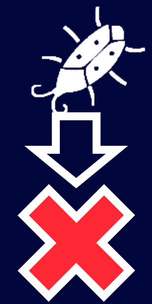
Validation program

```
void matgen (REAL a[], int lda, int n, REAL b[], REAL *norma)
{
    int init, i, j;
    init = 1325;
    *norma = 0.0;
    for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
            init = 3125*init % 0x100000000l;
            a[lda*j+i] = (init - 32768.0)/16384.0;
            *norma = (a[lda*j+i] > *norma) ? a[lda*j+i] : *norma;
        }
    }
    for (i = 0; i < n; i++) {
        b[i] = 0.0;
    }
    for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
            b[i] = b[i] + a[lda*j+i];
        }
    }
    return;
}

void dgefa(REAL a[], int lda, int n, int ipvt[], int *info){
    /*     internal variables       */
    REAL t;
    int j,k,kp1,l,nm1;

    /*     gaussian elimination with partial pivoting       */

    *info = 0;
    nm1 = n - 1;
    if (nm1 >=  0) {
        for (k = 0; k < nm1; k++) {
            kp1 = k + 1;

            /* find l = pivot index */
            l = idamax(n-k,&a[lda*k+k],1) + k;
            ipvt[k] = l;

            /* zero pivot implies this column already
               triangularized */

            if (a[lda*k+l] != ZERO) {
                /* interchange if necessary */

                if (l != k) {
                    t = a[lda*k+l];
                    a[lda*k+l] = a[lda*k+k];
                    a[lda*k+k] = t;
                }

                /* compute multipliers */
                t = -ONE/a[lda*k+k];
                dscal(n-(k+1),t,&a[lda*k+k+1],1);

                /* row elimination with column indexing */
                for (j = kp1; j < n; j++) {
                    t = a[lda*j+l];
                    if (l != k) {
                        a[lda*j+l] = a[lda*j+k];
                        a[lda*j+k] = t;
                    }
                    daxpy(n-(k+1),t,&a[lda*k+k+1],1,
                    &a[lda*j+k+1],1);
                }
            }
            else {
                *info = k;
            }
        }
    }

    ipvt[n-1] = n-1;
    if (a[lda*(n-1)+(n-1)] == ZERO) *info = n-1;
    checkExpected(a, ipvt, info);
    return;
}

void dgesl(REAL a[],int lda,int n,int ipvt[],REAL b[],int job )
{
    REAL t;
    int k,kb,l,nm1;

    nm1 = n - 1;
    if (job == 0) {
        /* job = 0 , solve  a * x = b
           first solve  l*y = b         */
        if (nm1 >= 1) {
            for (k = 0; k < nm1; k++) {
                l = ipvt[k];
                t = b[l];
                if (l != k){
                    b[l] = b[k];
                    b[k] = t;
                }
                daxpy(n-(k+1),t,&a[lda*k+k+1],1,&b[k+1],1 );
            }
        }

        /* now solve  u*x = y */
        for (kb = 0; kb < n; kb++) {
            k = n - (kb + 1);
            b[k] = b[k]/a[lda*k+k];
            t = -b[k];
            daxpy(k,t,&a[lda*k+0],1,&b[0],1 );
        }
    }
    else {
```

QED Trace

• • •

```
R1  ← R1  + 1
R18 ← R18 + 1
R1  == R18
```



R1	R18
3 ~~2~~	2

• • •

```
R1 ← 0
```

• • •

```
Check R1 == 0
```

Error masked!

No QED: bug escape

QED: quick detection

Diversity-Enhanced QED

Validation program

QED Trace

a ← 1; a' ← a; b ← 1, b' ← b

R1 ← a + b

R18 ← 5a' + 5b'

6 ~~2~~	R1
14 ~~10~~	R18

5×R1 == R18 ✗

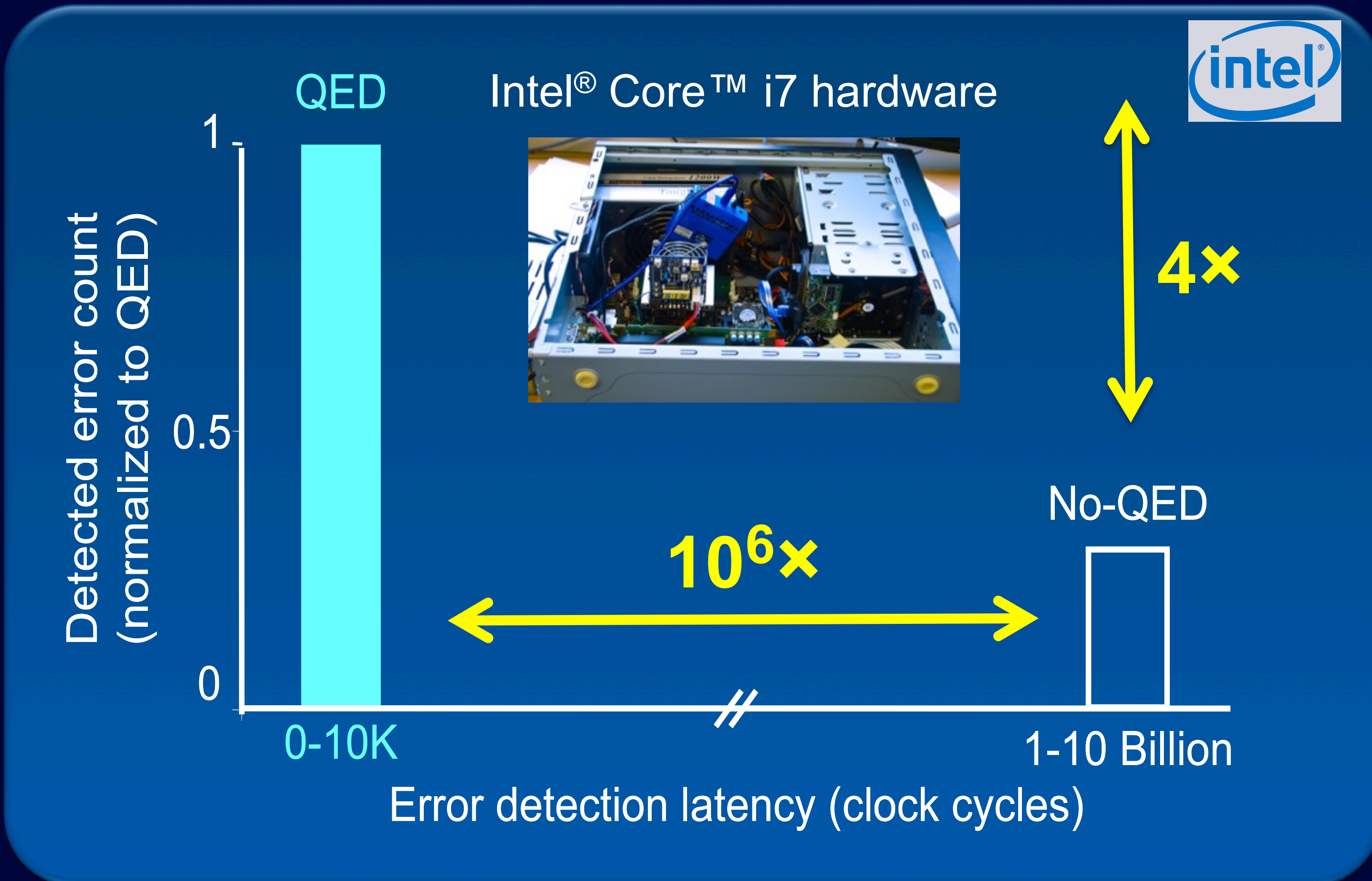
Many diversity techniques

e.g., ED^{4}I [Oh IEEE Trans. Comp. 02]

QED Coverage Considerations

- Challenge
  - Intrusiveness: coverage impact ?
- Systematic solutions
  - QED family tests
  - Hardware-enhanced QED

QED Effective for Electrical Bugs



[Lin IEEE TCAD 14]

E-QED

- **Electrical bug localization**

	E-QED
Flip-flop candidates	18 (1 million total) **50,000× localization**
Area impact	2.5% (actually 0%)
Debug effort	Automatic
Runtime	~ 9 hours

OpenSPARC T2 SoC (500M transistors)

[Singh CAV 17]

QED Effective for Logic Bugs

Freescale SoC hardware

Random Instruction Test Generator

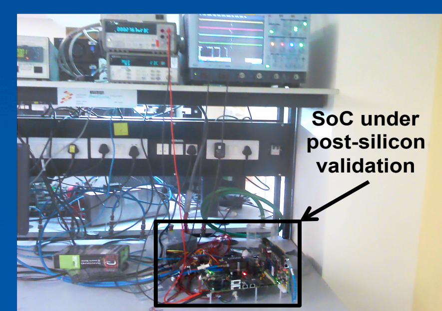
Core 0 | Core 1 | Core 2 | Core 3 | • • • | Core N

Interconnection network

Shared Caches | Memory Controllers | Accelerators | Other uncore components

Error detection latency (cycles)

No QED	QED
15 Billion	**9**



[Lin IEEE TCAD 14]

Symbolic QED

- Pre-silicon
  - Logic bugs

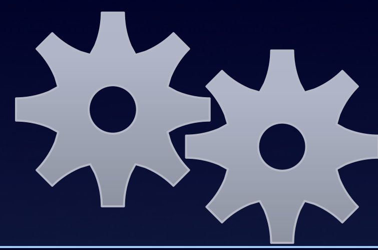
[Lin ITC 15, Singh IEEE TCAD 18]

Traditional Bounded Model Checking

Property

Design

BMC Tool



If property violated

Counter-example = bug trace

Traditional BMC vs. Symbolic QED

	Traditional BMC	Symbolic QED
Properties	Manual	**Automatic QED checks (Universal property)**
Design size	Small blocks	**Large SoCs**
False fails	Possible	**None**

BMC Symbolic QED

"Universal" Property +
QED-consistent starting state

Design + QED Module
(no hardware overhead)

BMC Tool

If property violated

Counter-example = bug trace

[Lin ITC 15, Singh IEEE TCAD 18]

Symbolic QED Results

- Many designs: processors, accelerators, …

Infineon case study

Several automotive microcontroller cores

All recorded bugs detected (+ more)
20 seconds or less per bug
Verification productivity: 8× – 60× improved

[Singh DATE 19]

BUT…

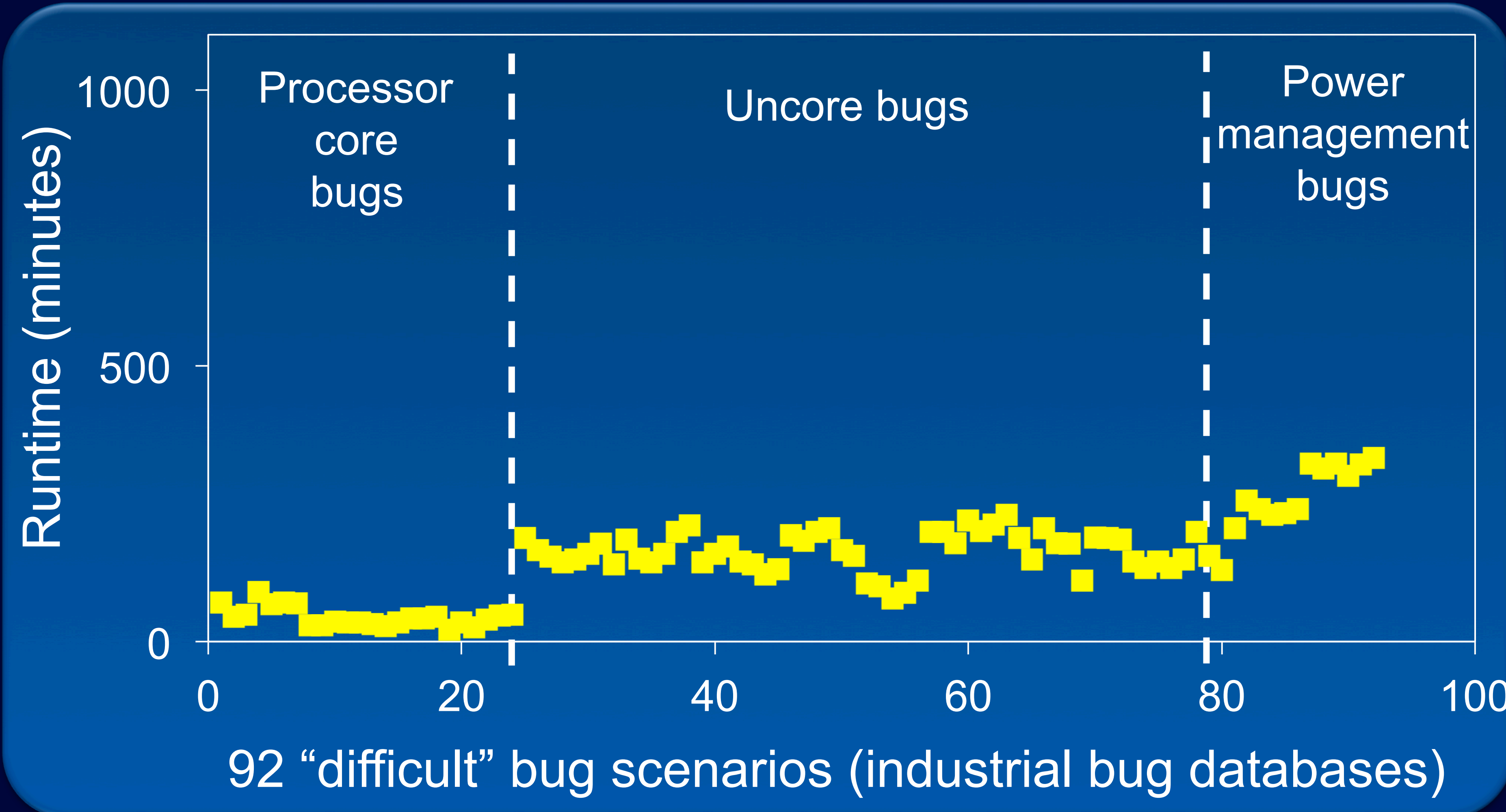
- Big designs ?

- Solution: QED checks compositional
  - Preserved across partial instances

[Lin ITC 15, Singh IEEE TCAD 18]

Symbolic QED: Billion-Transistor SoCs

20 mins. to 7 hours



OpenSPARC T2 SoC (500M transistors): difficult bugs inserted

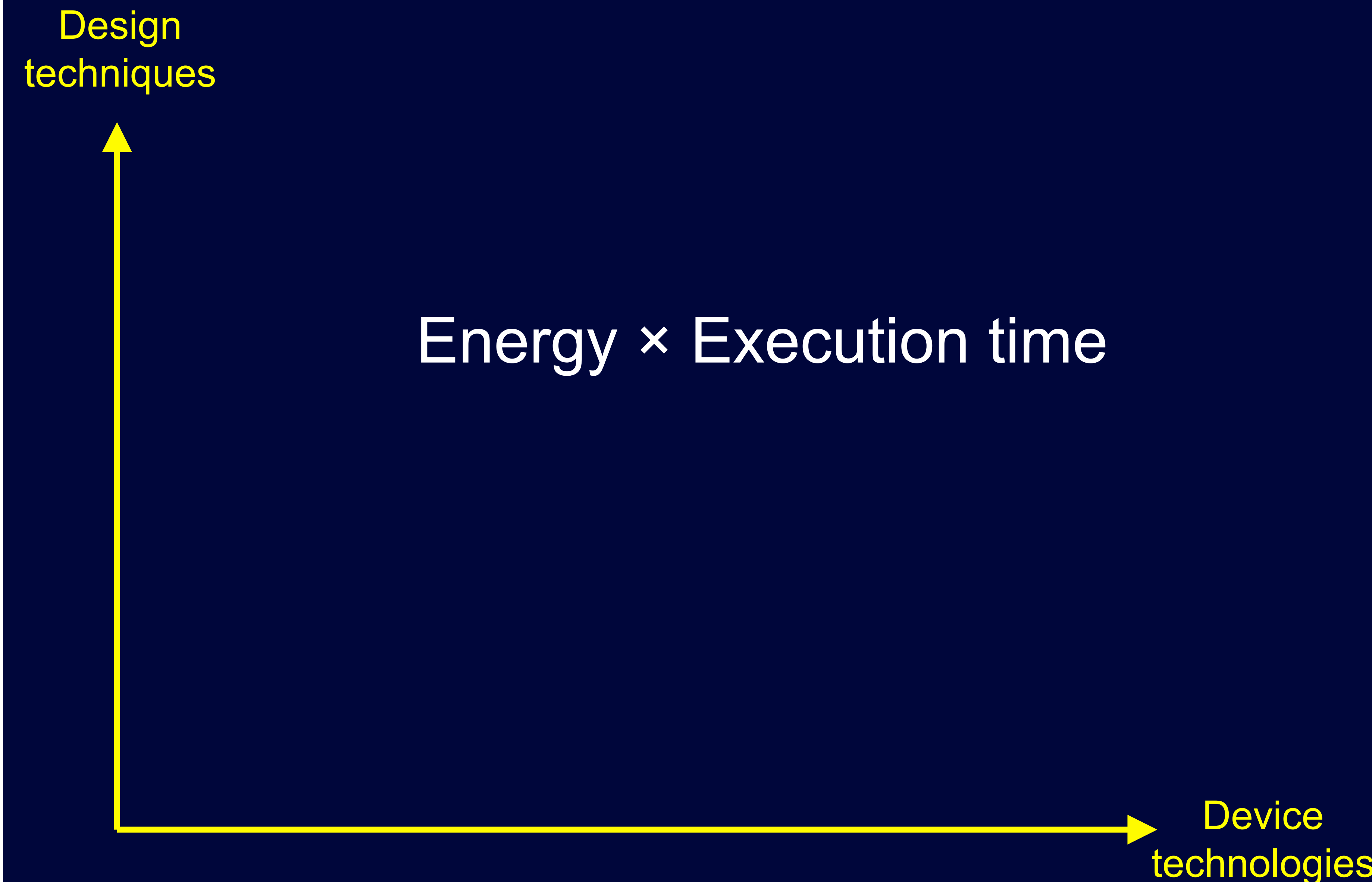
More Opportunities

- Hardware security
  - Derive new vulnerabilities
    - Beyond Spectre, Meltdown
  - Detect Trojans
- Firmware
- Large-scale systems

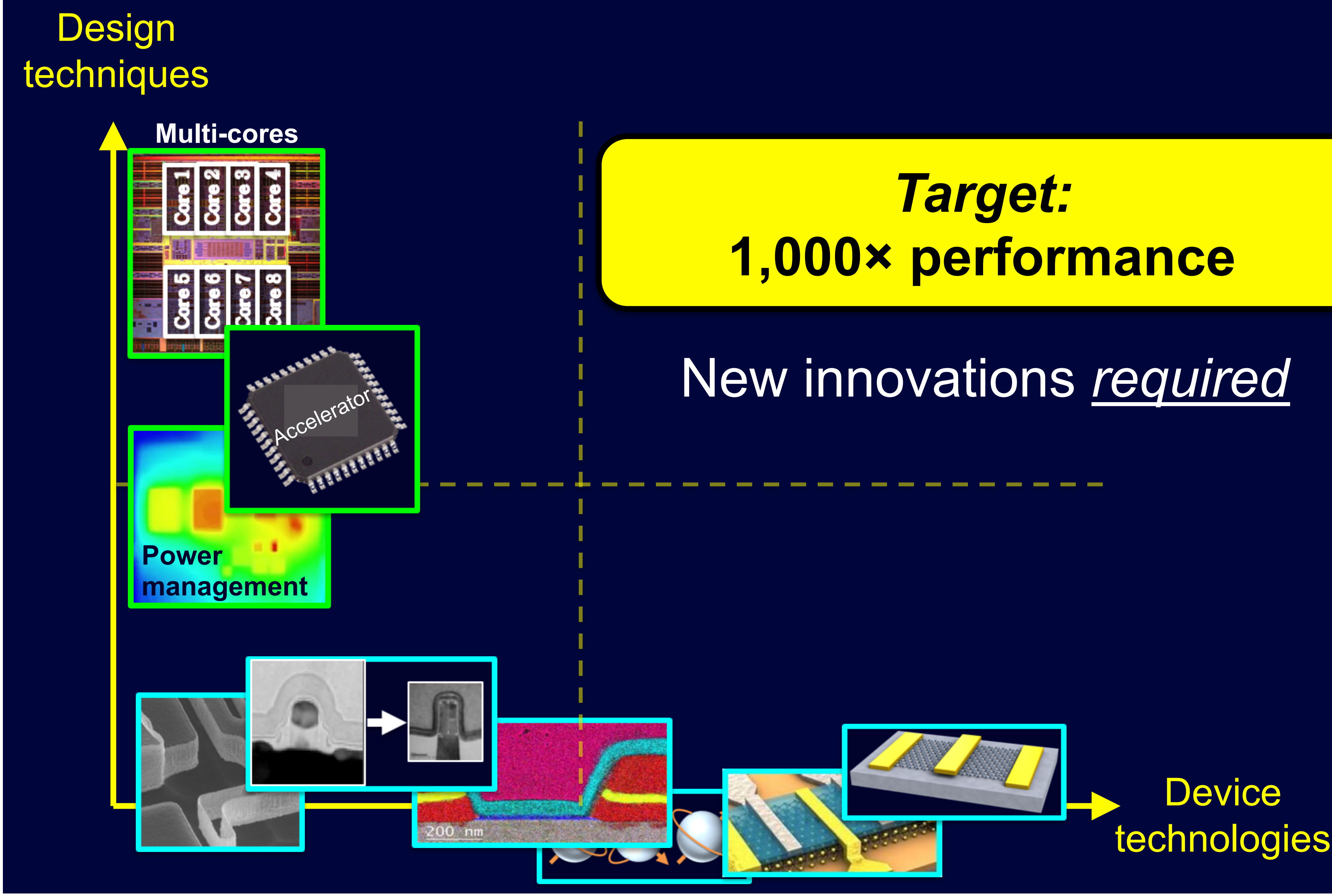
Outline

- NanoSystems: *N3XT 1,000×*

Improve Computing Performance



Improve Computing Performance



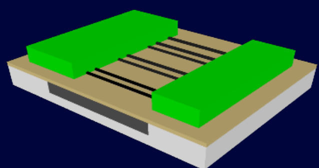
Solution: Nano*<u>Systems</u>*

Transform new nanotech

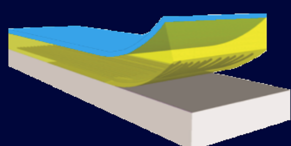
into new systems

enable new applications

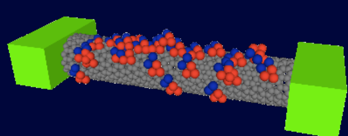
New devices



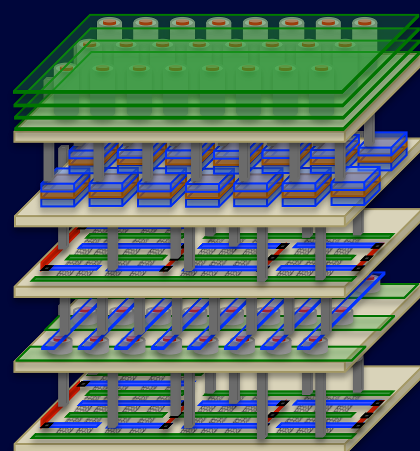
New fabrication



New sensors



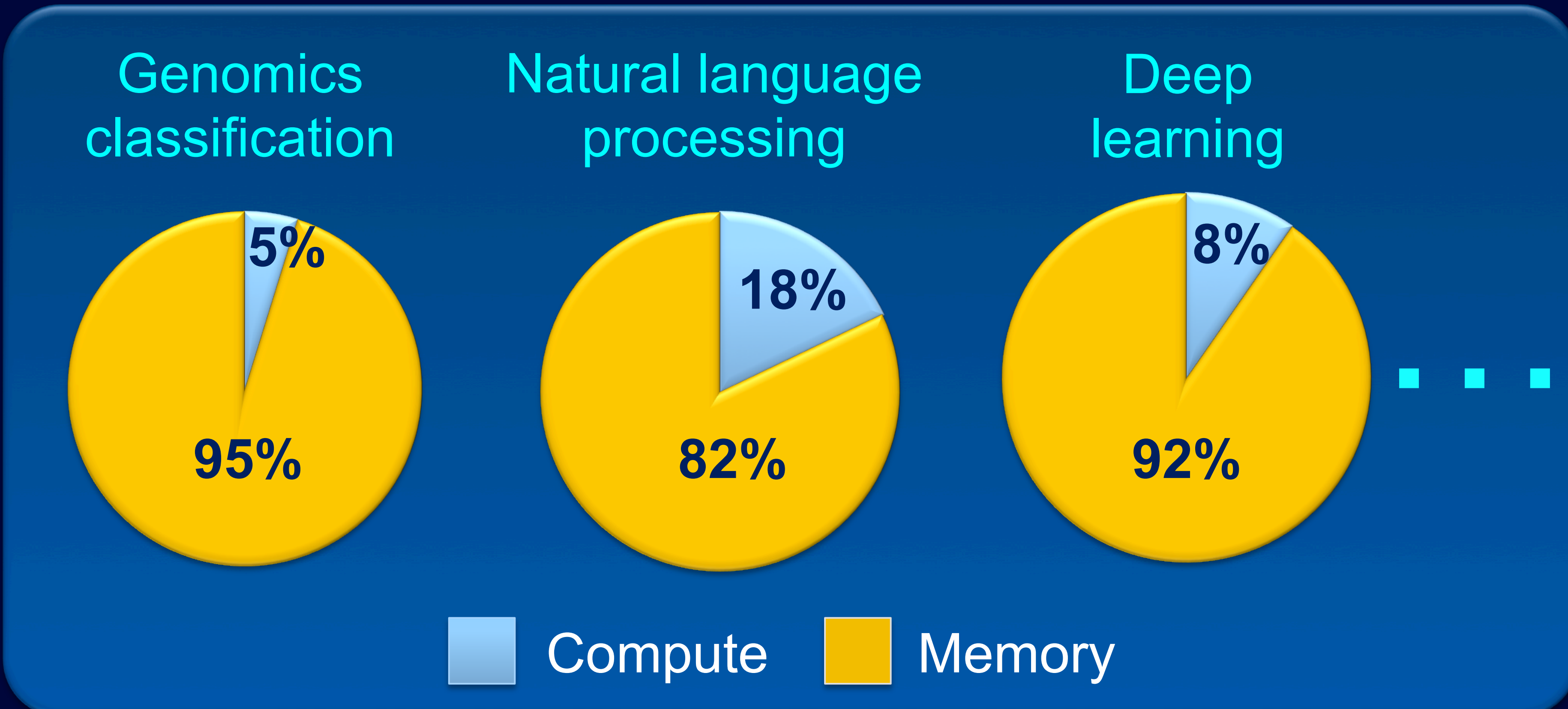
New architectures



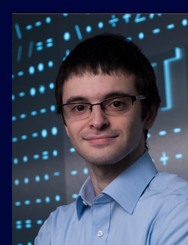
Abundant-Data Applications

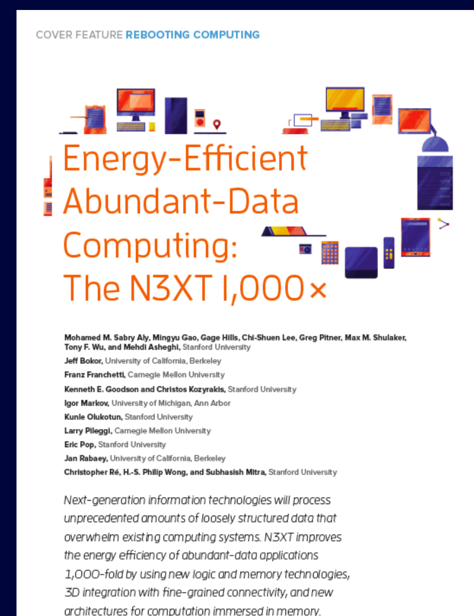
Memory wall: processors, accelerators

Wide range of domains



Nano-Engineered Computing Systems Technology





COVER FEATURE REBOOTING COMPUTING

Energy-Efficient Abundant-Data Computing: The N3XT 1,000×

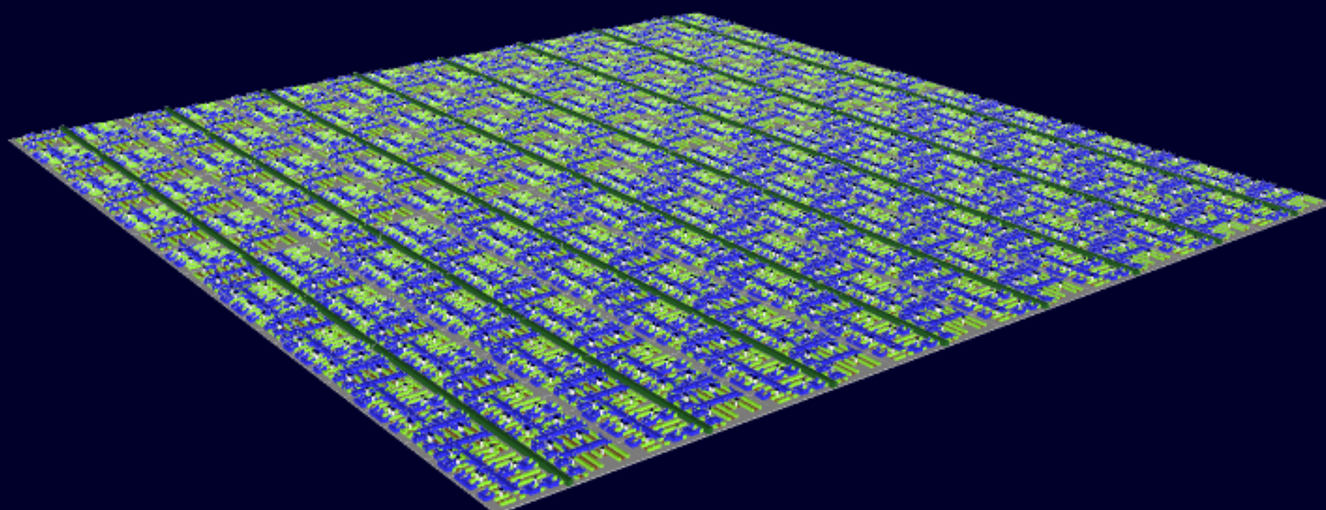
Mohamed M. Sabry Aly, Mingyu Gao, Gage Hills, Chi-Shuen Lee, Greg Pitner, Max M. Shulaker, Tony F. Wu, and Mehdi Asheghi, Stanford University
Jeff Bokor, University of California, Berkeley
Franz Franchetti, Carnegie Mellon University
Kenneth E. Goodson and Christos Kozyrakis, Stanford University
Igor Markov, University of Michigan, Ann Arbor
Kunle Olukotun, Stanford University
Larry Pileggi, Carnegie Mellon University
Eric Pop, Stanford University
Jan Rabaey, University of California, Berkeley
Christopher Ré, H.-S. Philip Wong, and Subhasish Mitra, Stanford University

Next-generation information technologies will process unprecedented amounts of loosely structured data that overwhelm existing computing systems. N3XT improves the energy efficiency of abundant-data applications 1,000-fold by using new logic and memory technologies, 3D integration with fine-grained connectivity, and new architectures for computation immersed in memory.

[Aly IEEE Computer 15, Proc. IEEE 19]

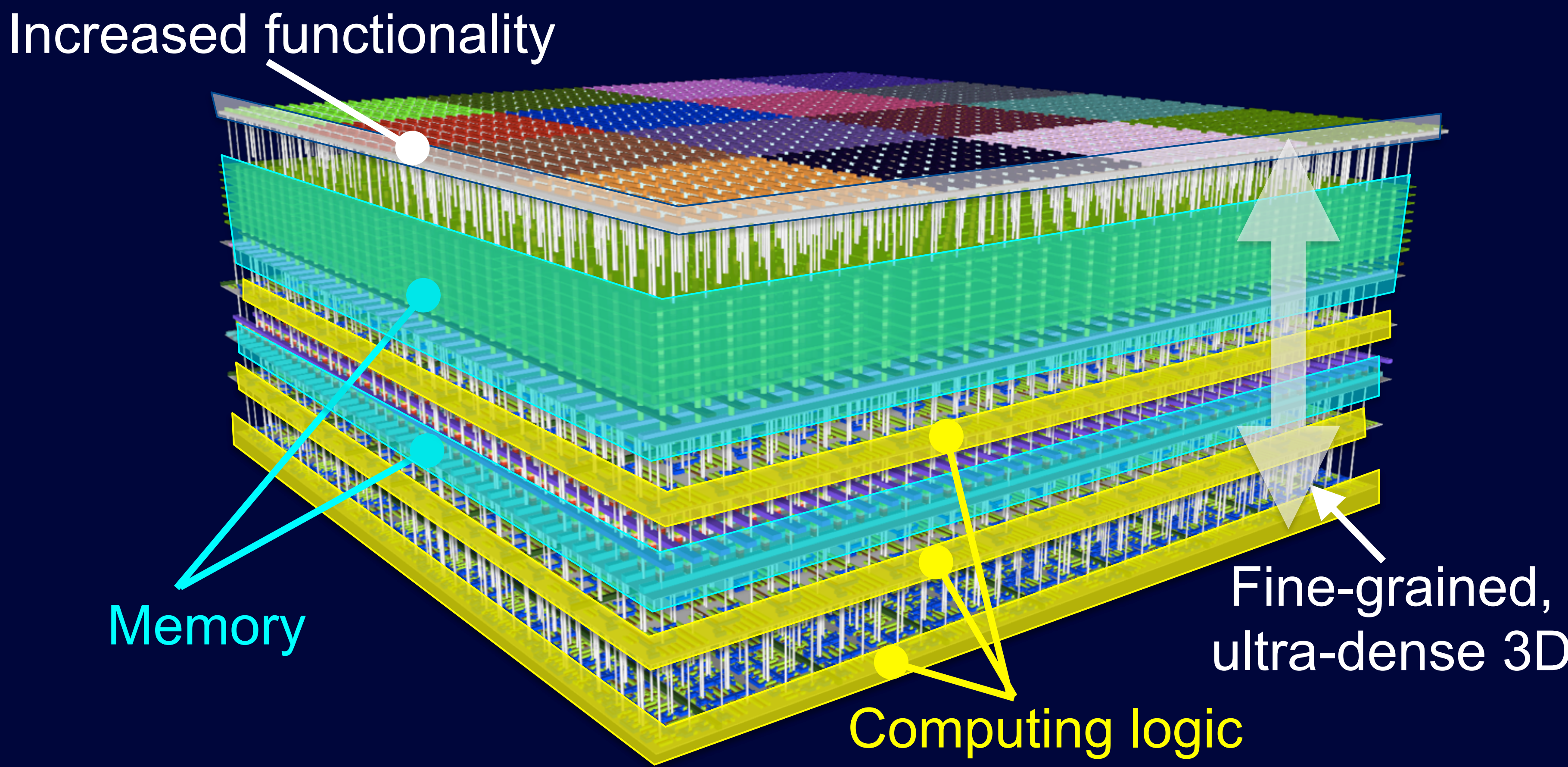
N3XT NanoSystems

Computation immersed in memory



N3XT NanoSystems

Computation immersed in memory



Impossible with today's technologies

N3XT Computation Immersed in Memory

3D Resistive RAM
Massive storage

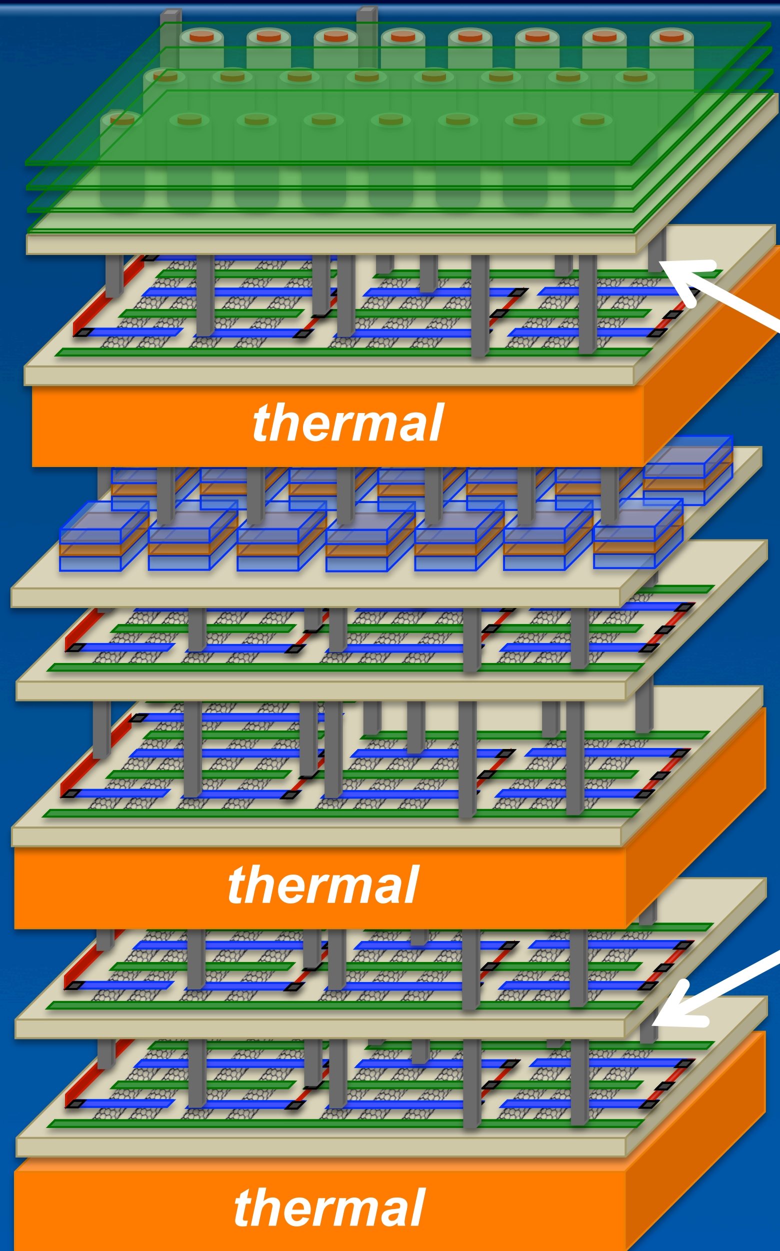
⋮

1D CNFET, 2D FET
Compute, RAM access

STT MRAM
Quick access

1D CNFET, 2D FET
Compute, RAM access

1D CNFET, 2D FET
Compute, Power, Clock

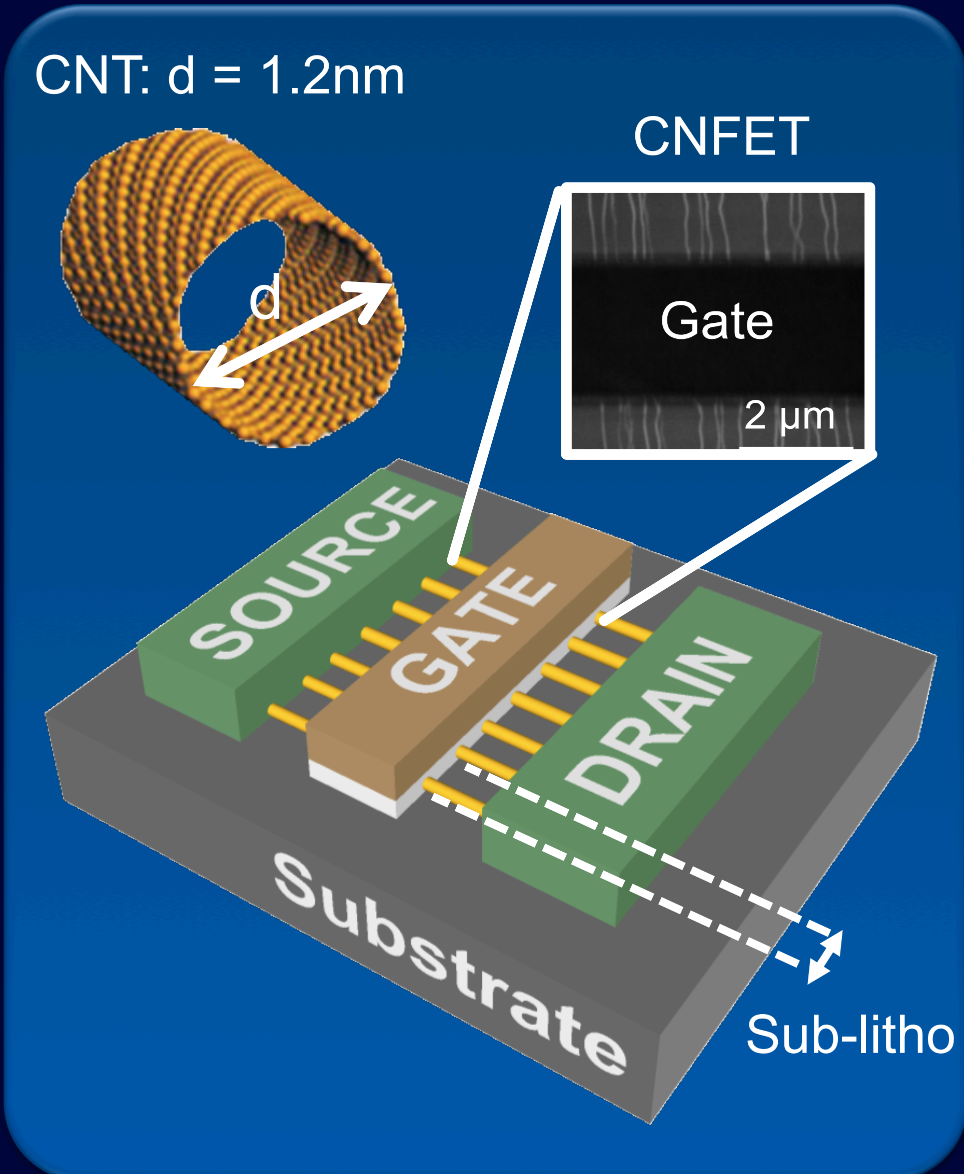


No TSV

Ultra-dense, fine-grained vias

Silicon compatible

Carbon Nanotube FET (CNFET)



Energy **D**elay **P**roduct

- ~ 10× benefit

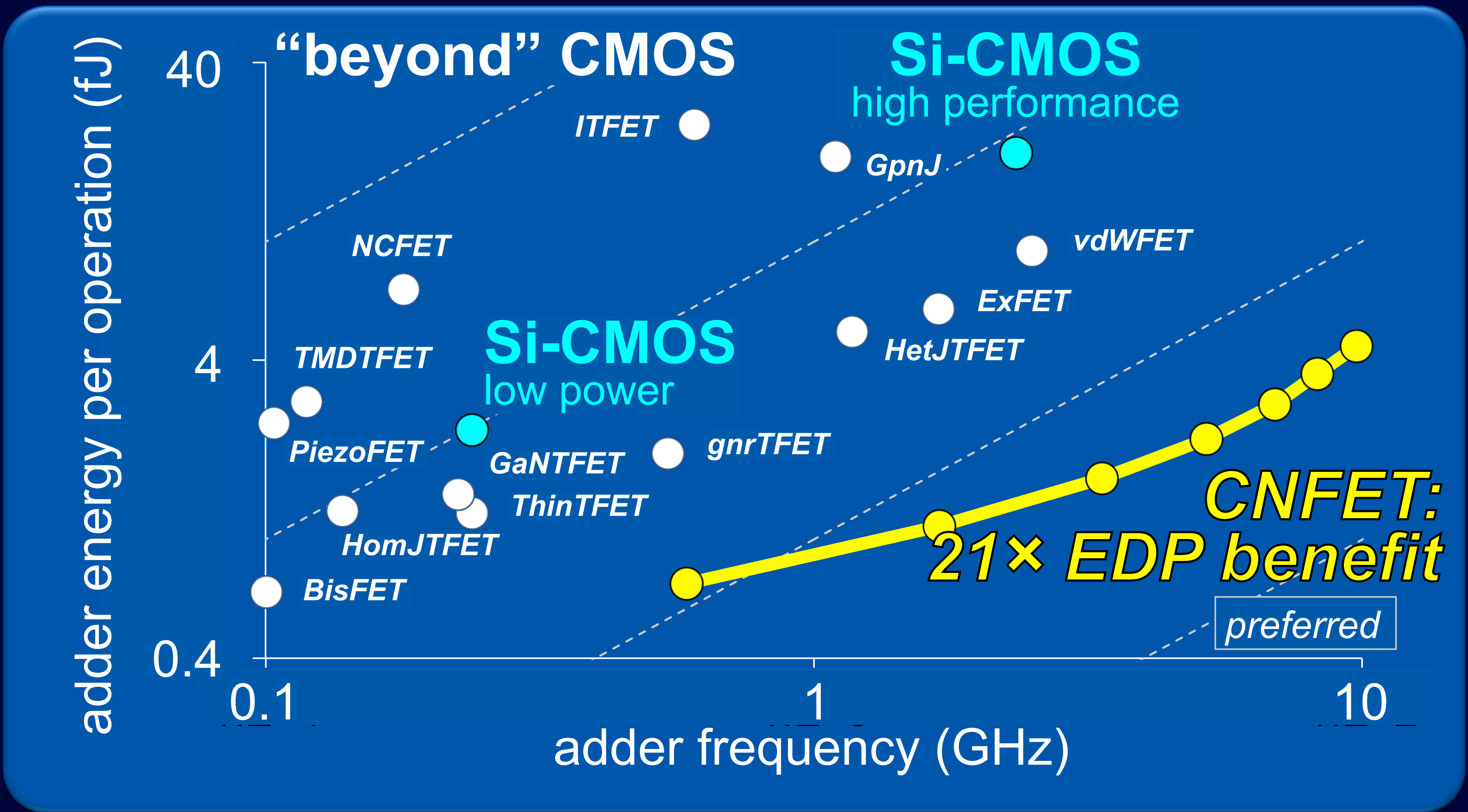
Processor case studies

[Stanford + IMEC + TSMC, IBM, other commercial]

[Hills IEEE TNANO 18] Stanford + IMEC + TSMC

Putting into Perspective

- Existing technology benchmarking **+ CNFETs**

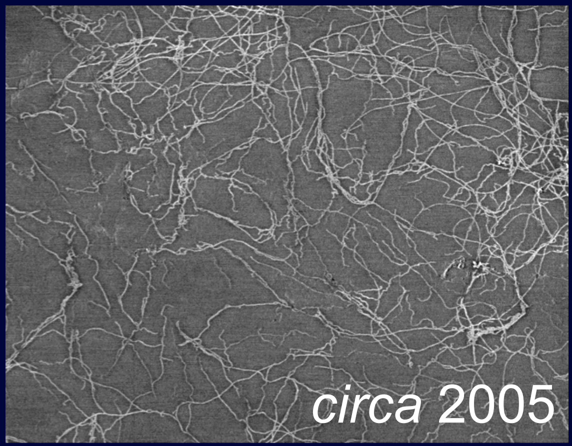


32-bit adder [Nikonov & Young, 2013 & 2015]

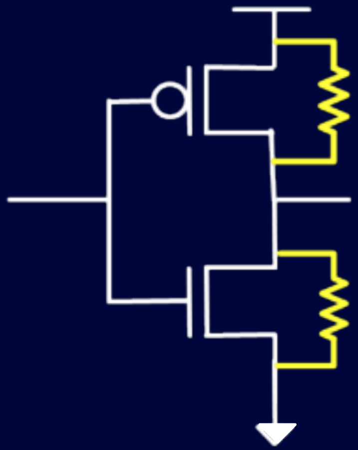
Big Promise, Major (Past) Obstacles

- Process advances alone inadequate

Mis-positioned CNTs



Metallic CNTs

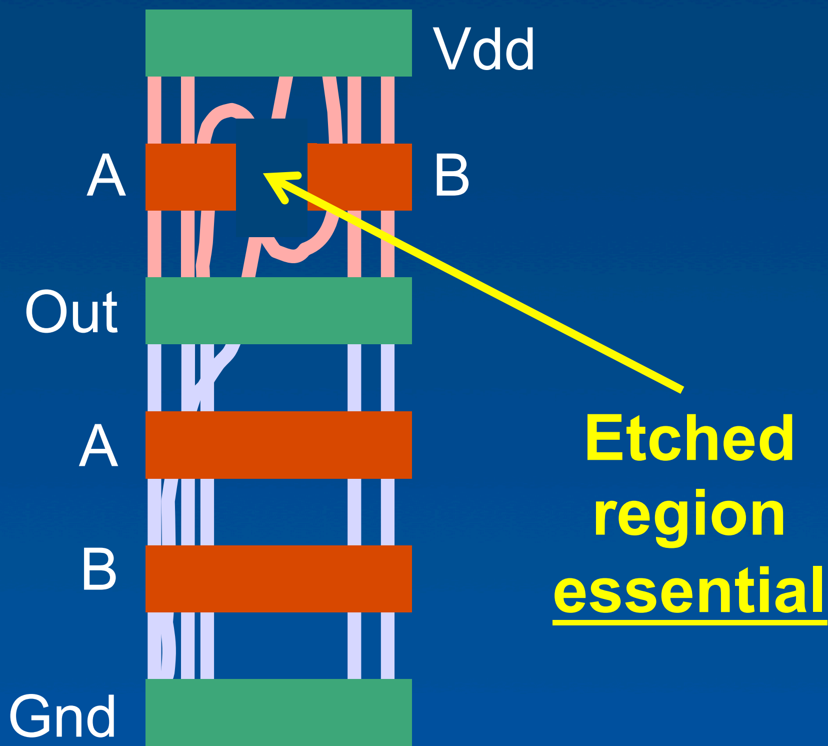


Imperfection-immune paradigm

[Zhang IEEE TCAD 12]

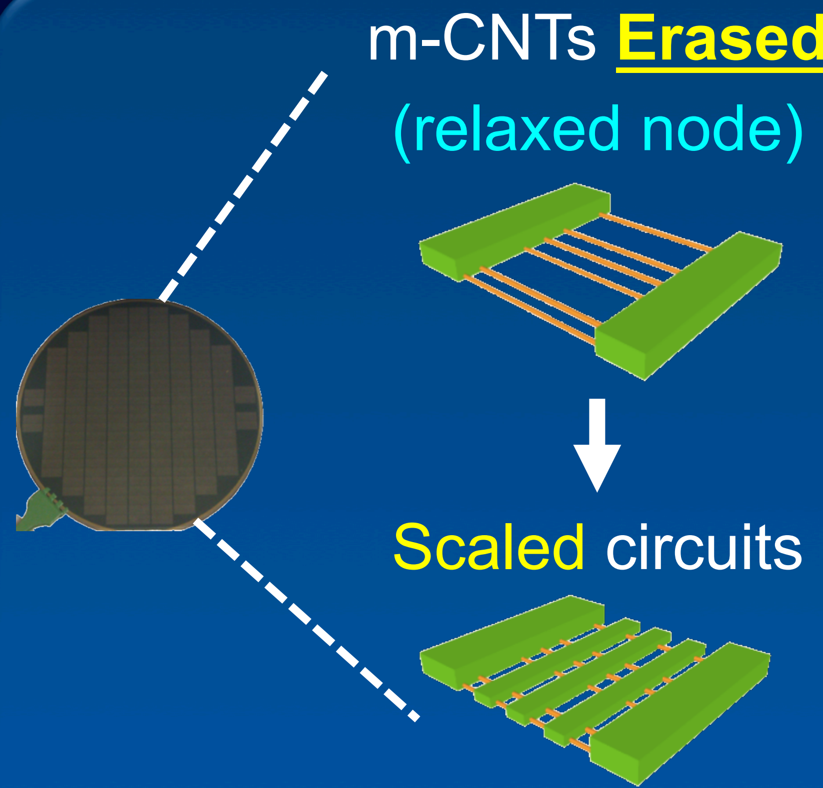
Imperfection-Immune VLSI

Mis-positioned CNTs



- Arbitrary logic functions

Metallic CNTs



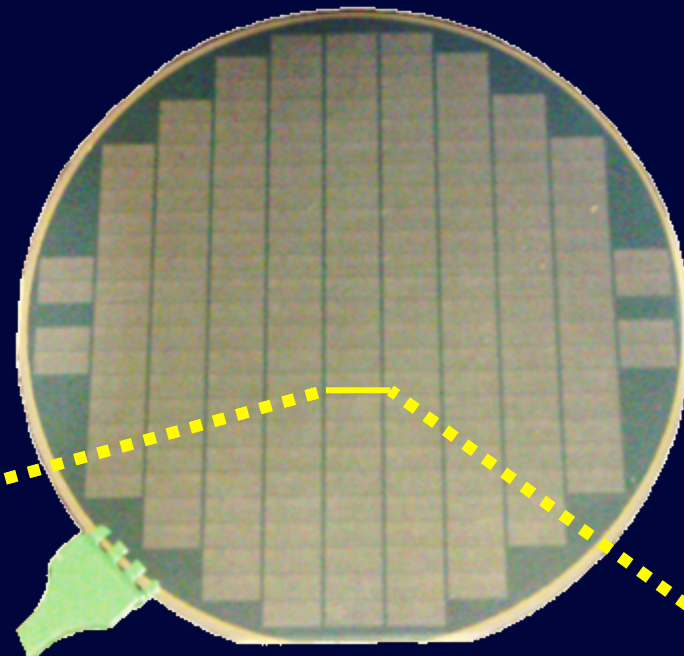
- Scalable m-CNT Removal

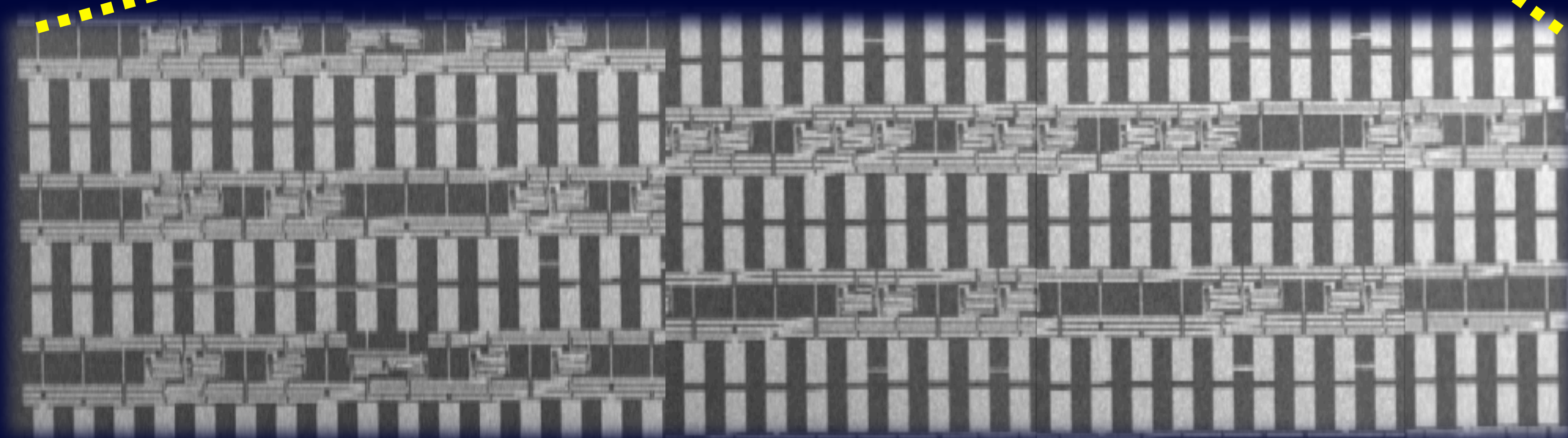
[Patil Symp. VLSI Tech. 08, TCAD 09, Shulaker IEDM 15]

Most Importantly

- VLSI processing
  - No per-unit customization
- VLSI design
  - Immune CNT library

CNT Computer





[Shulaker Nature 13]

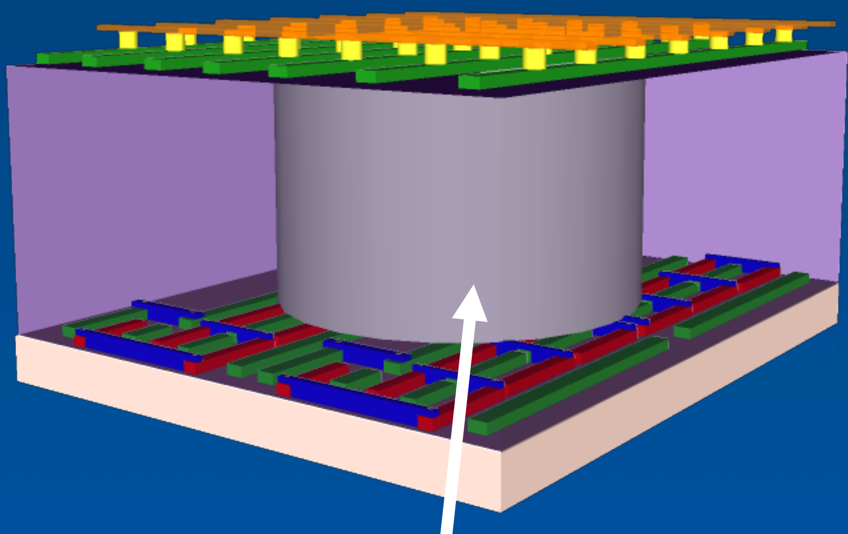
10X EDP, BUT…

How can we do better ?

3D Integration

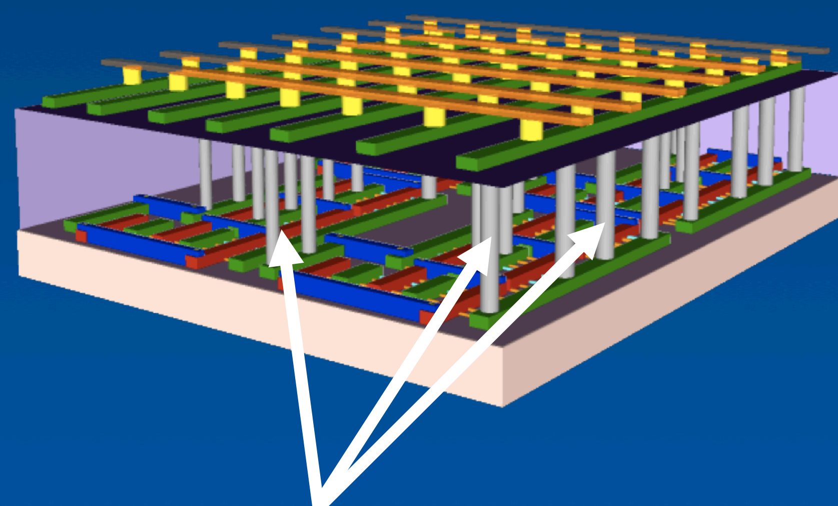
- Massive ILV density >> TSV density

TSV (chip stacking)



Through silicon via (TSV)

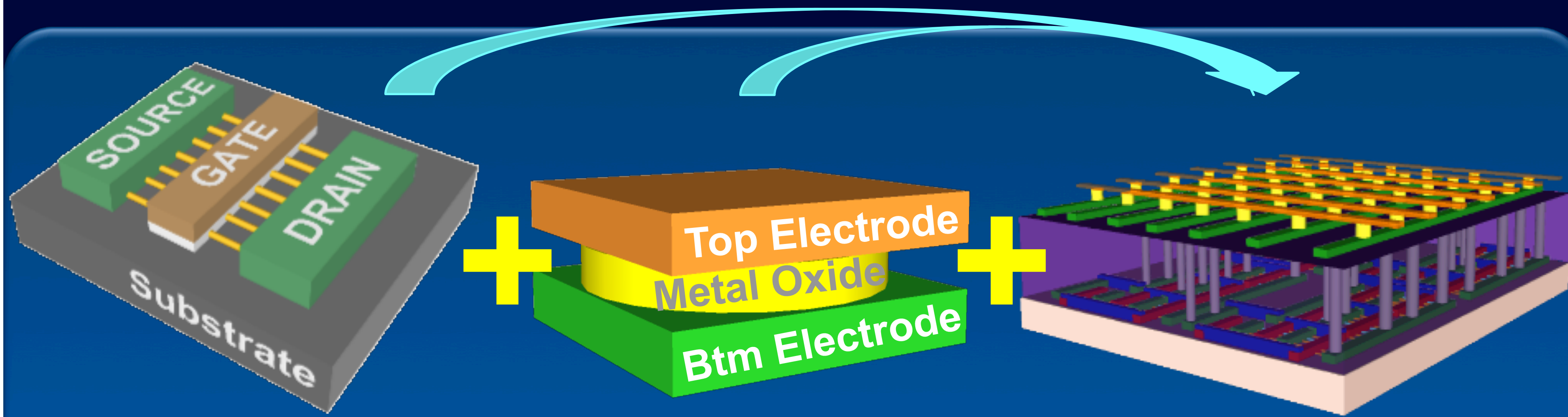
Dense, e.g., monolithic



Nano-scale inter-layer vias (ILVs)

Device + Architecture Benefits

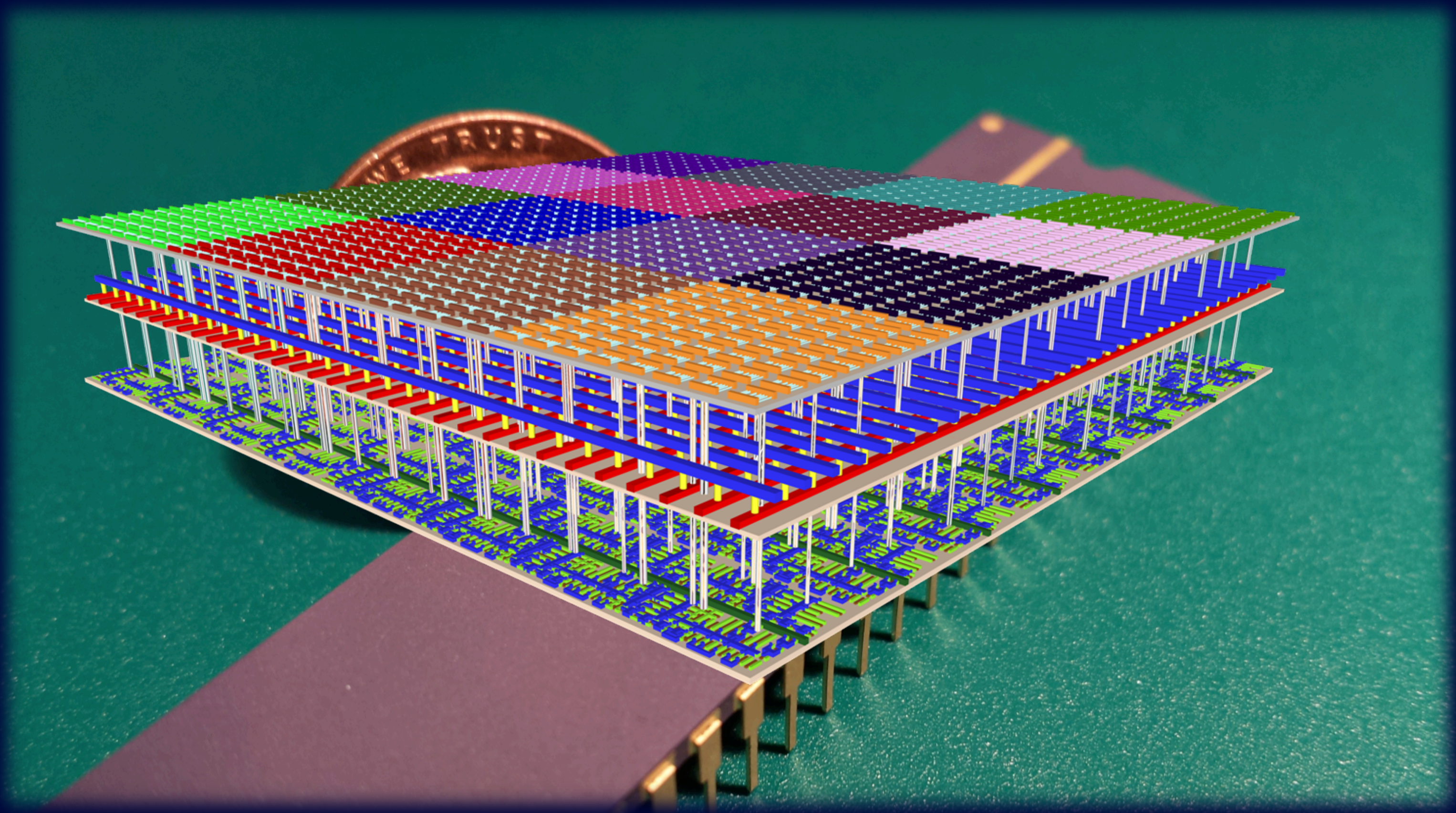
Naturally enabled: < 400 °C fabrication



Emerging logic + Emerging memory + Monolithic 3D

3D NanoSystem

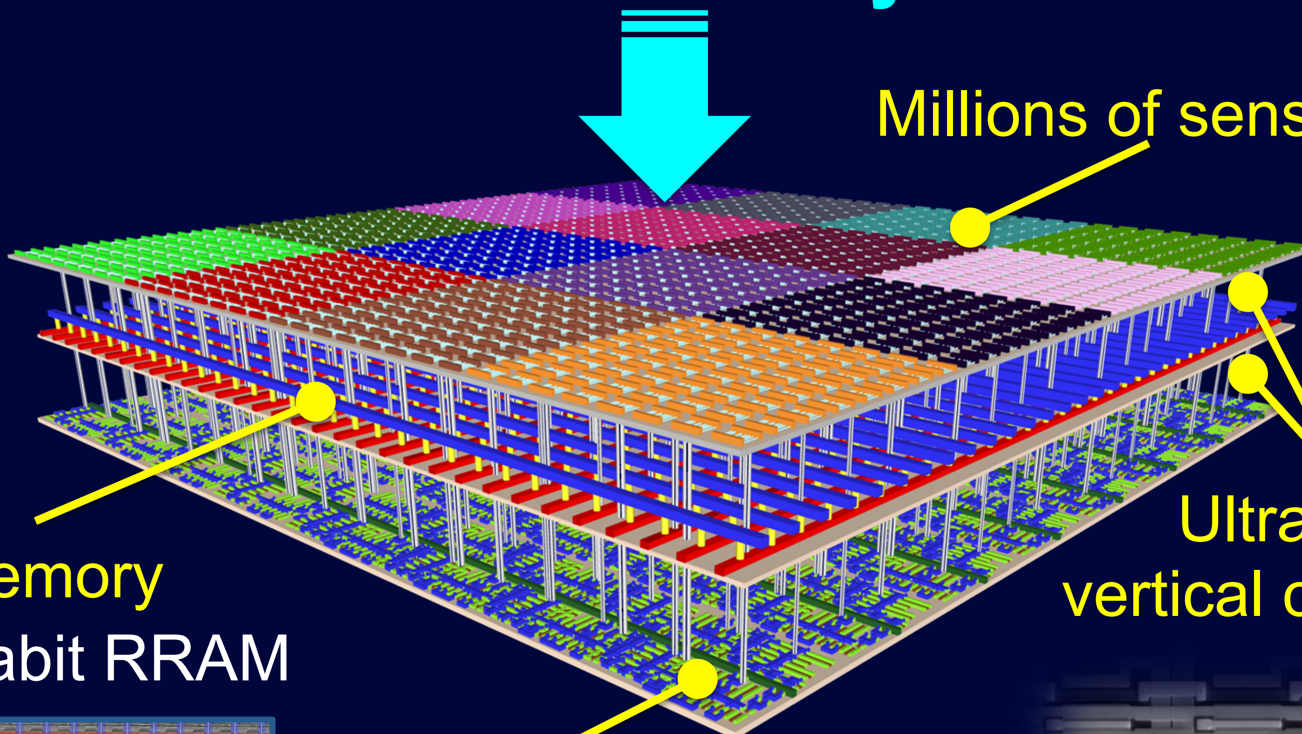
2 Million CNFETs, 1 Mbit RRAM

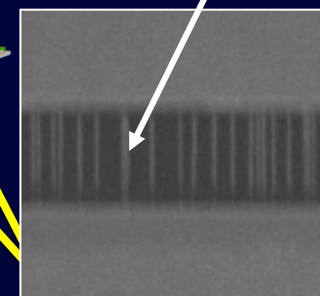


[Shulaker Nature 17]

3D NanoSystem

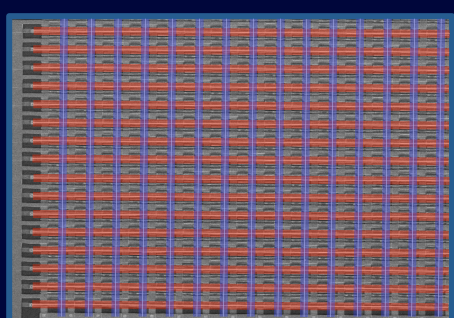
Abundant data: Terabytes / second





Ultra-dense vertical connections

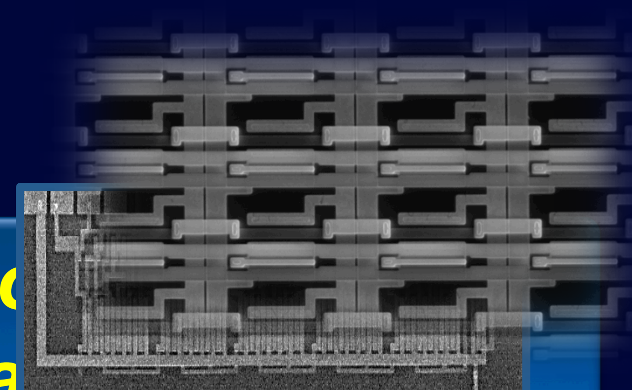
1 Megabit RRAM



lassification accelerator

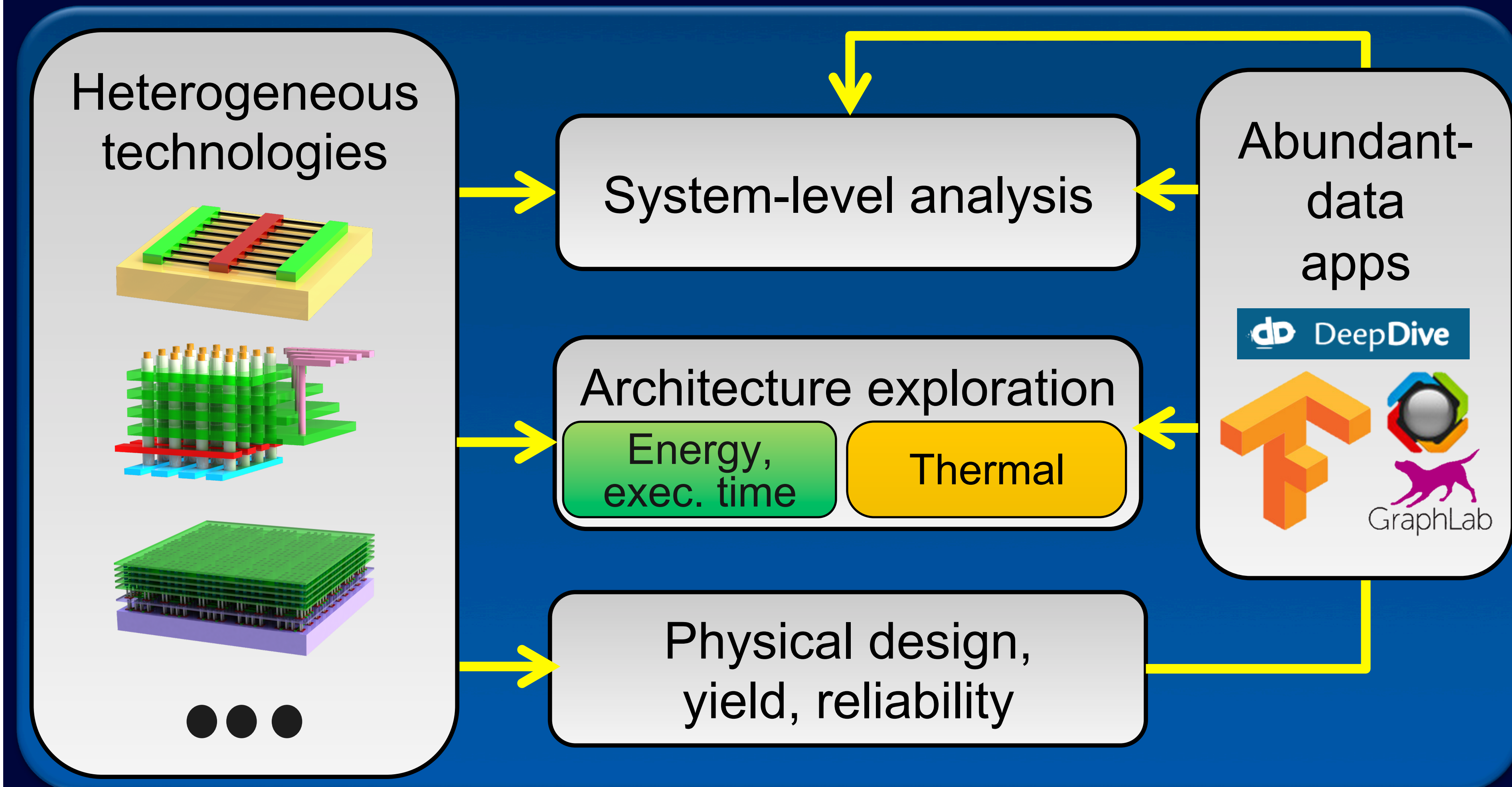
In-situ classificatic

Extensive, accura



N3XT Simulation Framework

Joint technology, design & app. exploration

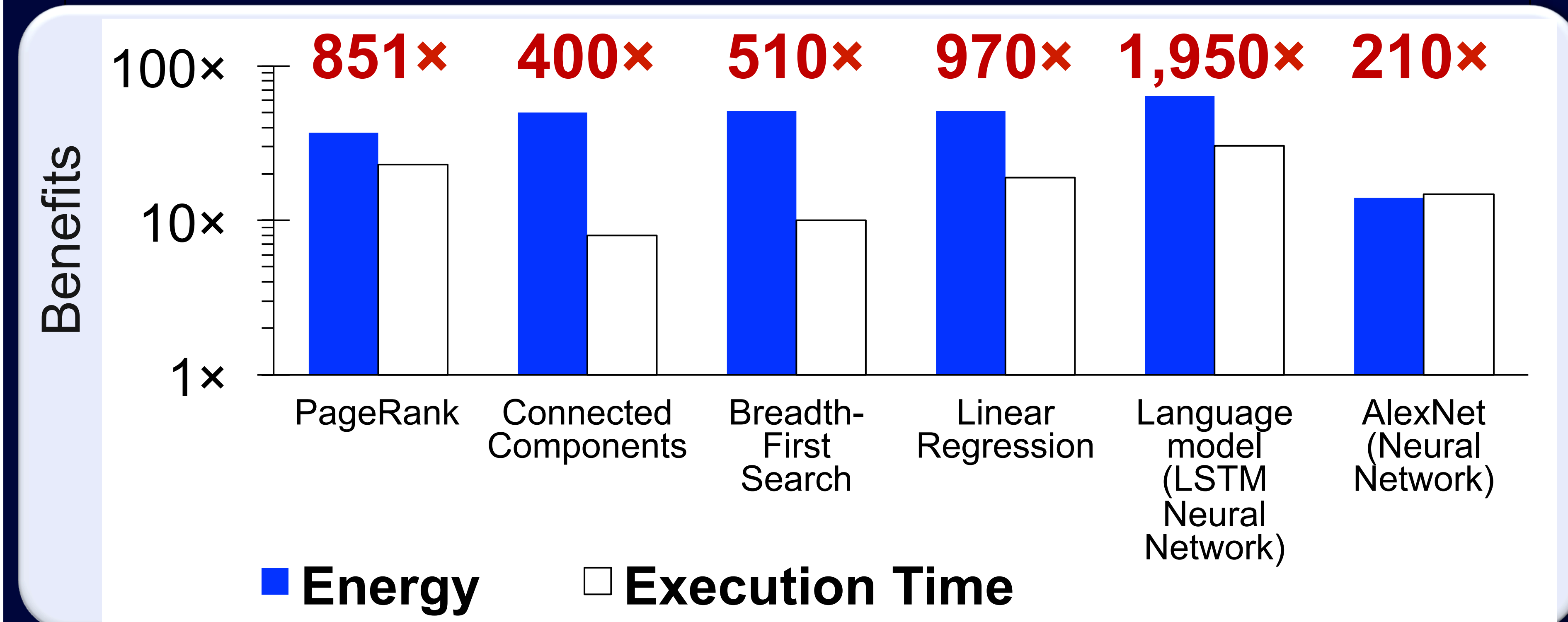


[Sabry Proc. IEEE 19]

Massive Benefits: Deep Learning, Graph Analytics, …

IBM graph analytics

~*1,000*× benefits, existing software



[Sabry Proc. IEEE 19]

Many NanoSystem Opportunities

Cross-layer solutions

Tech. + design + app.

1. Multiple bits / RRAM cell

Special RRAM programming + weight encoding

2. New RRAM resilience

10-year continuous inference

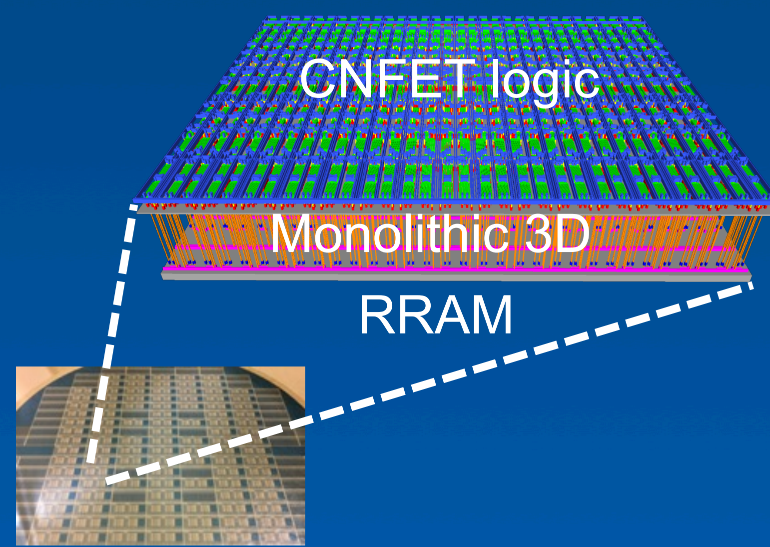
[Le IEEE TED 19, Wu ISSCC 19]
Stanford + CEA LETI

Brain-inspired ⊃ neural nets

e.g., Hyperdimensional (HD)

1. One-shot learning

2. Classify language: 98% accuracy

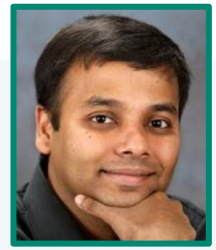


Live ISSCC demo

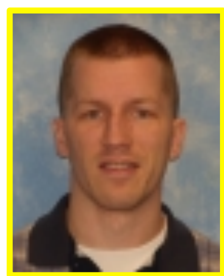
[Wu ISSCC 18, IEEE JSSC 18]
Stanford + Berkeley

DARPA 3DSoC Program


Max Shulaker
Anantha Chandrakasan



Subhasish Mitra
H.-S. Philip Wong
Simon S. Wong

Brad Ferguson
Mark Nelson

Jefford Humes

Outline

- Conclusion

Neural Interfaces: Natural Resolution

1. Treat disorders, augment capabilities

2. Understand brain

3. New neuro-inspired computing ?

New abstractions, new ICs, closed-loop brain experiments

Wu Tsai Neurosciences Institute
Stanford University

Collaborator: Prof. E.J. Chichilnisky, Stanford

Conclusion

- Robust operation, performance, new applications
  - Big opportunities
  - New solutions: elegantly simple, effective

QED & Symbolic QED	*N3XT*
Pre-silicon & post-silicon	NanoSystems
Automatic, overnight	Ultra-dense & fine-grained 3D
Billion-transistor scale	*1,000*× opportunity