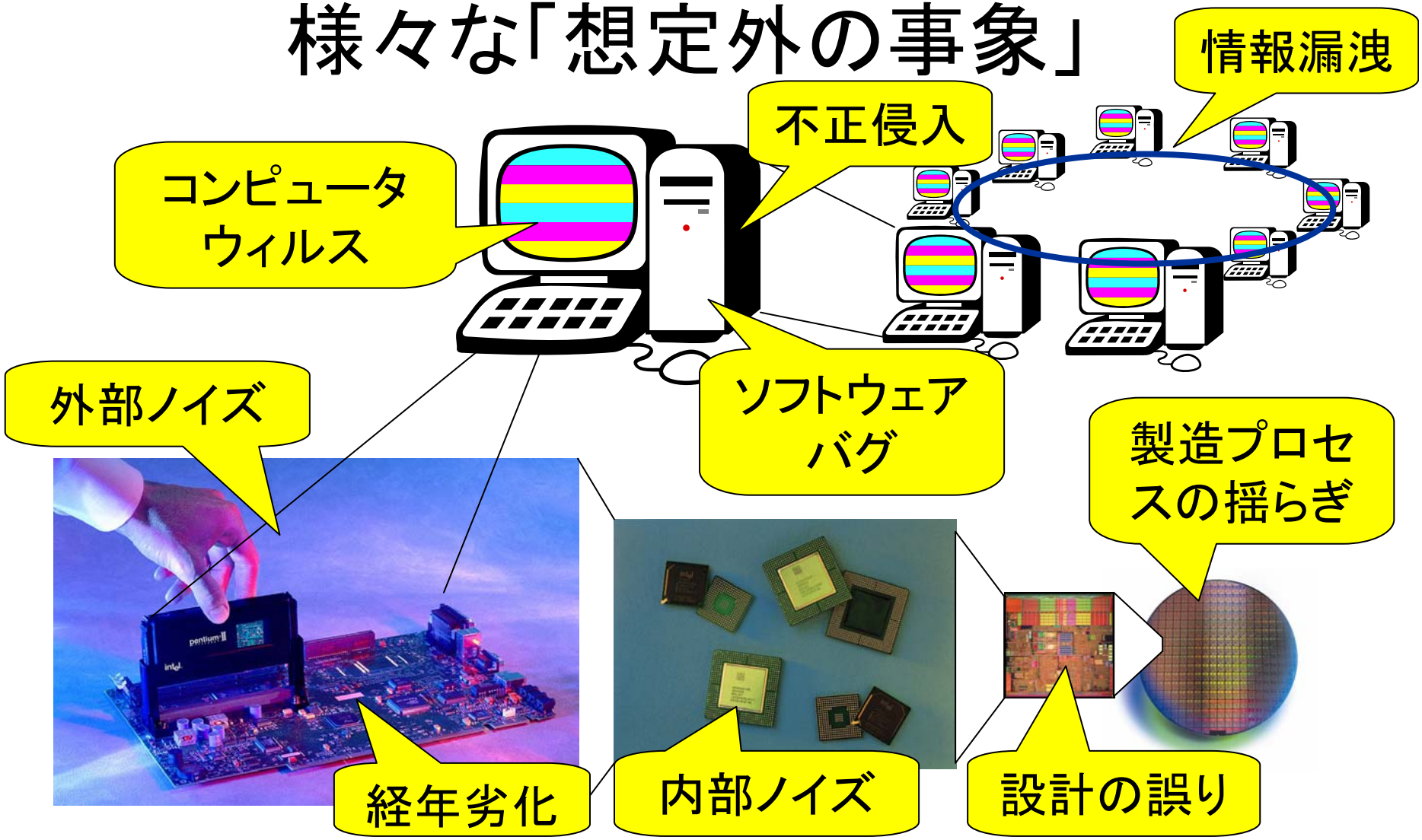
ディペンダブル・プロセッサの研究動向

九州大学 大学院システム情報科学研究院

井上弘士（いのうえこうじ）

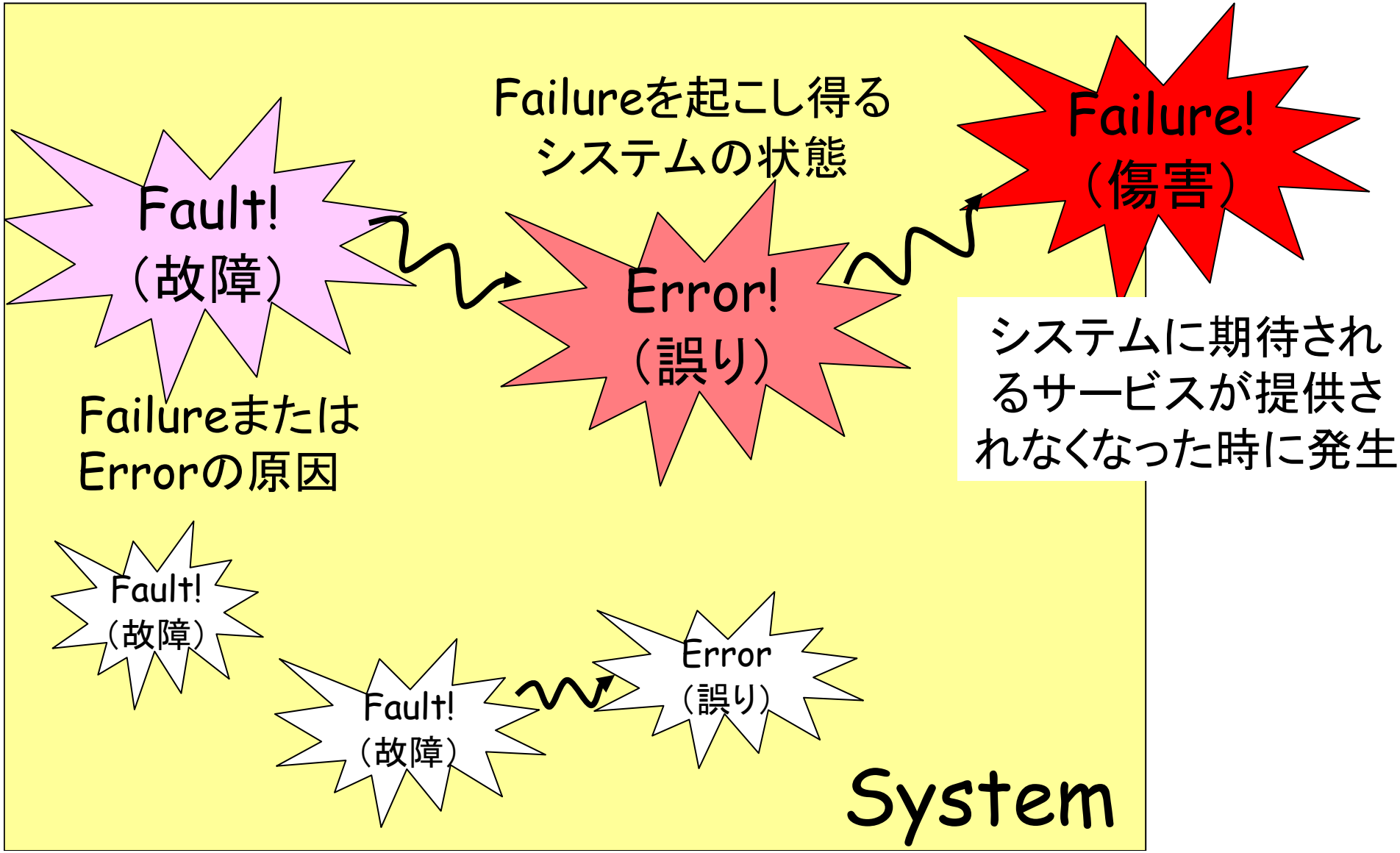
inoue@i.kyushu-u.ac.jp

コンピュータ・システムにおける様々な「想定外の事象」



テクノロジの進歩に伴い「想定外事象」の発生は増加

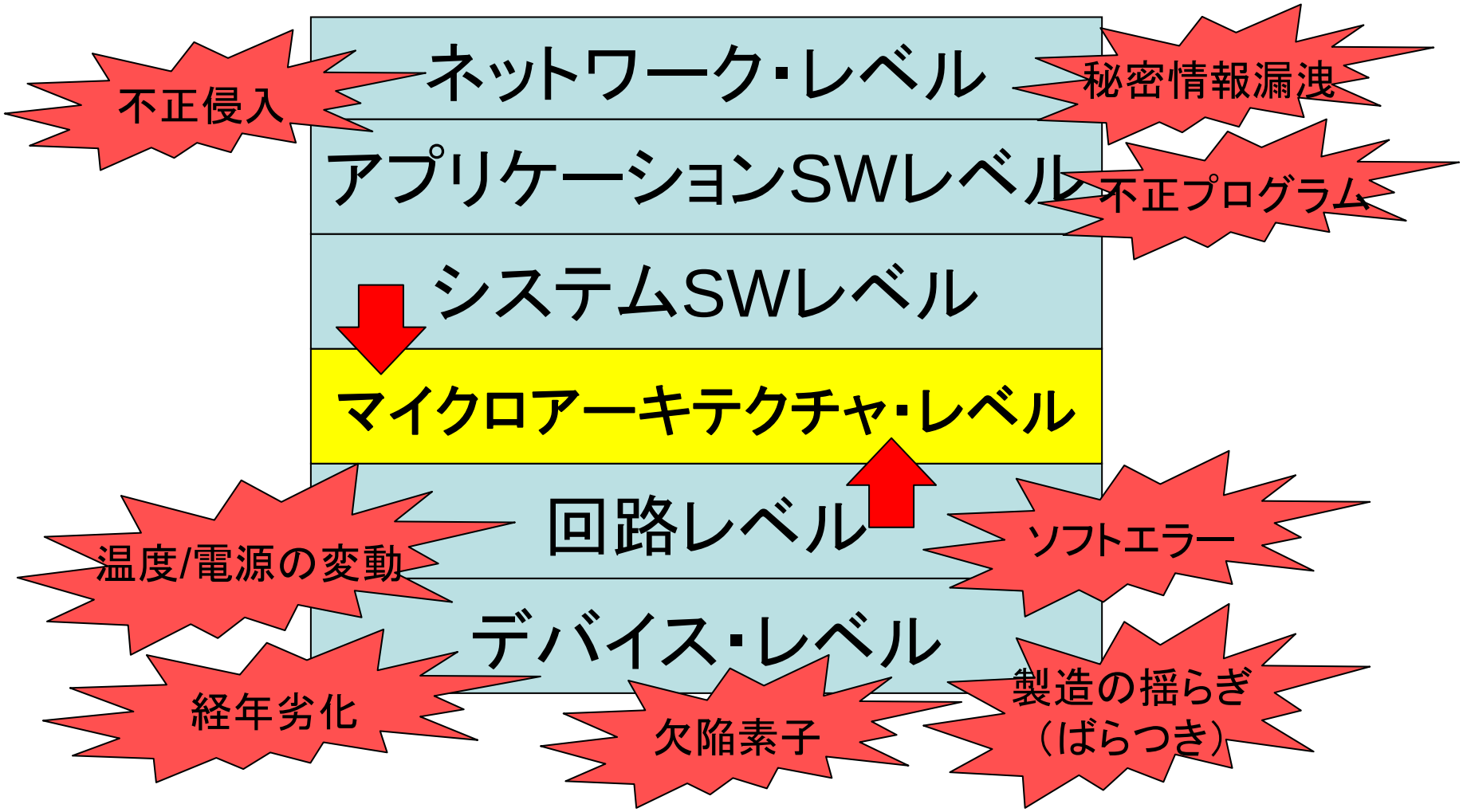
ディペンダビリティ阻害要因の因果関係



ディペンダブル・マイクロプロセッサ

- マイクロプロセッサのお仕事は？
  - 「正しいプログラム」を「正しく実行」する
    - 正しい命令の取得（フェッチ）と実行の繰り返し
    - 必要に応じてメモリにデータを正しく書込み/読出し
- マイクロプロセッサにとっての「想定外事象」は？
  - 正しく実行できない！
    - ハードウェア（回路）が正しく動作しない
    - 誰かに実行を邪魔される
  - 正しいプログラムではない！
    - 不正なプログラムを実行させられる

なぜ，アーキテクチャ・レベル？



チュートリアル内容

- 信頼性（耐故障性）を向上する！
  - ソフトエラー対策
  - タイミングエラー対策
- 安全性を向上する！
  - 不正プログラムの実行防止
  - メモリデータ改ざんの防止
  - 情報漏えいの防止
- プロセスばらつきの影響を緩和する！
  - 製造後のパイプライン・チューニング
- ハードウェアバグを回避する！
  - ハードウェア・パッチング
- まとめ

チュートリアル内容

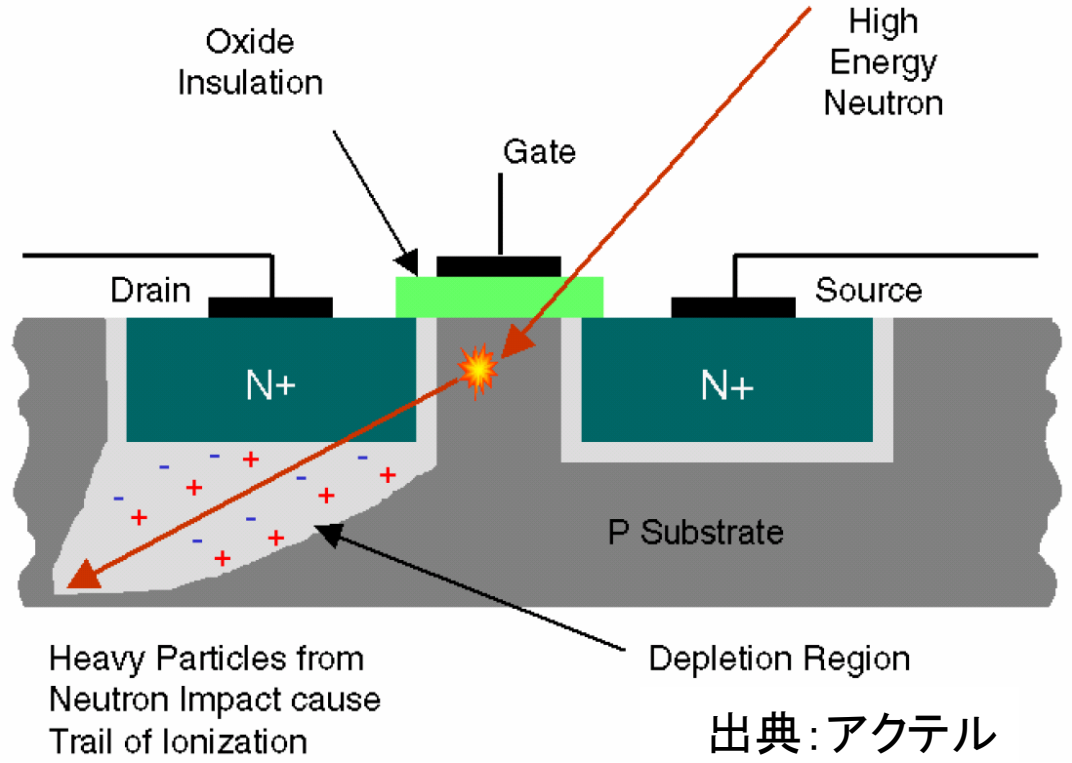
- 信頼性（耐故障性）を向上する！
  - ソフトエラー対策
  - タイミングエラー対策
- 安全性を向上する！
  - 不正プログラムの実行防止
  - メモリデータ改ざんの防止
  - 情報漏えいの防止
- プロセスばらつきの影響を緩和する！
  - 製造後のパイプライン・チューニング
- ハードウェアバグを回避する！
  - ハードウェア・パッチング
- まとめ

ソフトエラーとは？

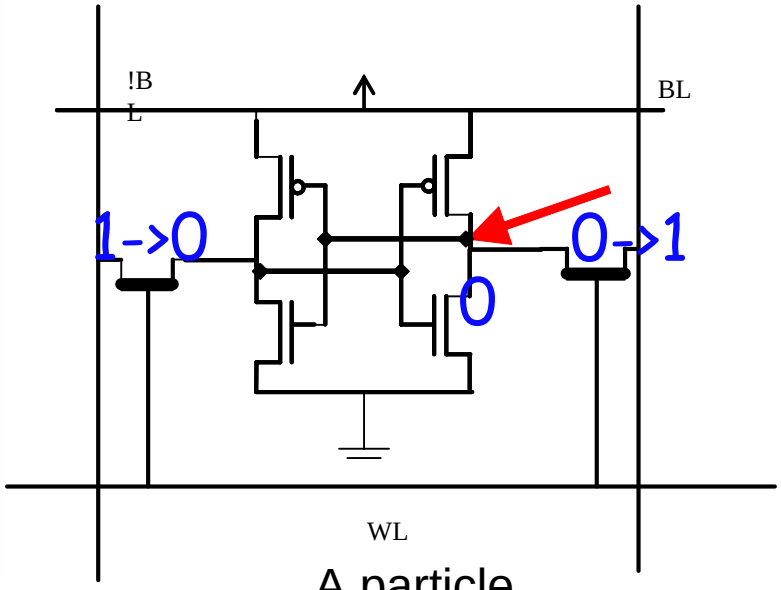
- 広義には・・・
  - 回路中で発生する一過性のエラー（誤り）
  - 回路そのものに致命的なダメージを与えるハードエラーとは異なり一時的に発生
  - Signal Integrity Errors, Thermally-induced Errorsなど

- 狭義には・・・
  - 外部放射線粒子によって引起こされる回路中の一過性エラー（誤り）
    - アルファ線や中性子線など
  - 回路内部で「アップセット・イベント」を引き起こす（値の反転）
  - Single Event Upset: SEU, ビット反転, ビットエラー

ソフトエラー



出典：アクテル

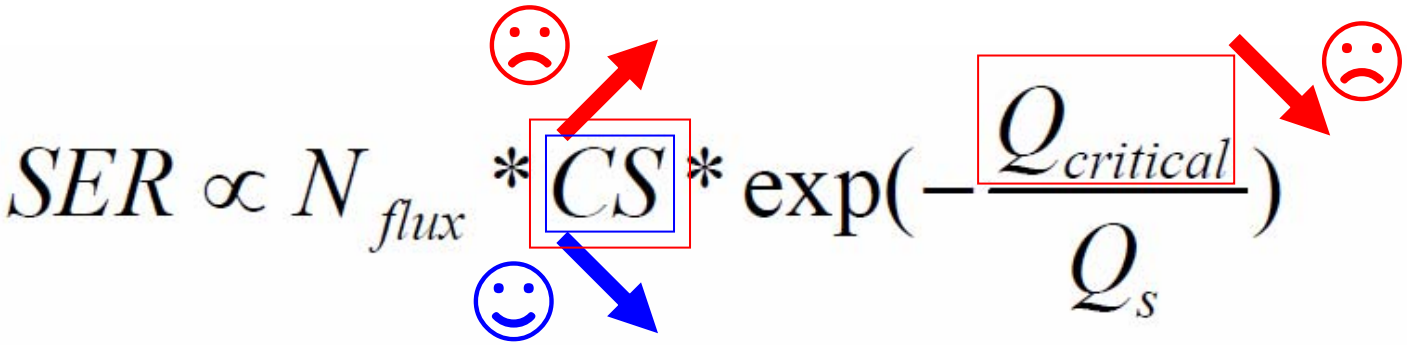


A particle strike

- OFF状態であったトランジスタに電流が流れる
- メモリに記憶した値が反転！

微細化が続くと・・・

「チップ全体」では増加　　ノード容量と電源電圧に比例



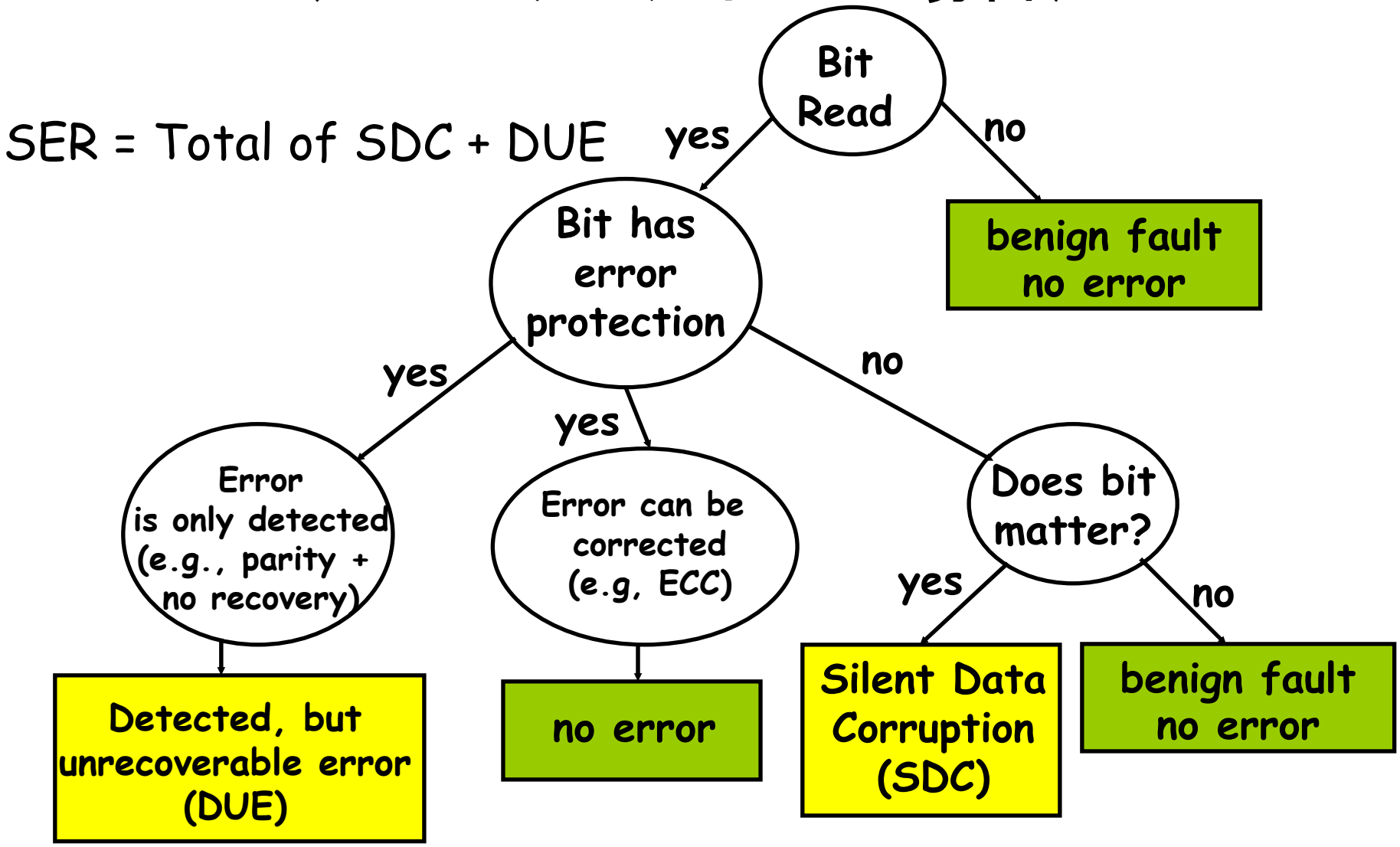
$$SER \propto N_{flux} * CS * \exp\left(-\frac{Q_{critical}}{Q_s}\right)$$

「ノード当たり」では削減

- N_{flux} : intensity of the neutron flux.
- CS : the area of the cross section of the node.
- $Q_{critical}$: critical charge necessary for a bit flip.
- Q_s : the charge collection efficiency.

「良性のFault」と「悪性のFault」（レジスタファイルの場合）

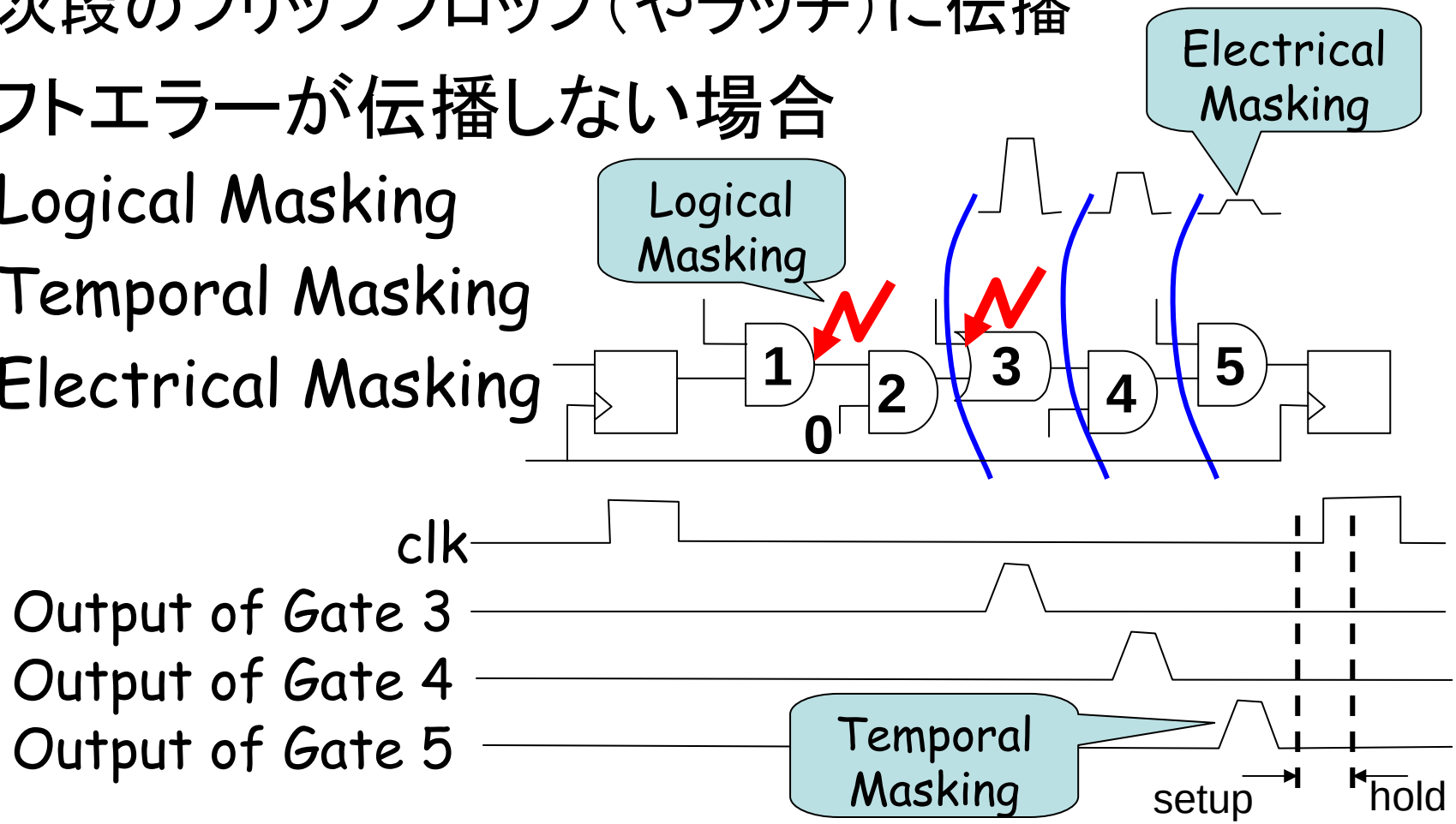
SER = Total of SDC + DUE



S. S. Mukherjee et al., "A Systematic Methodology to Compute the Architectural Vulnerability Factors of a High-Performance Microprocessor," Int'l Symp. On Microarchitecture, 2003.

ソフトエラーのマスキング（in Logic）

- 組合せ回路でのソフトエラー
  - 次段のフリップフロップ（やラッチ）に伝播
- ソフトエラーが伝播しない場合
  - Logical Masking
  - Temporal Masking
  - Electrical Masking



ソフトエラーのマスキング（Architectural State）

- 実行結果に影響しない命令の実行
  - Performance enhancing instructions – prefetch
  - NOPs
  - Wrong-path instructions
  - Dead instructions

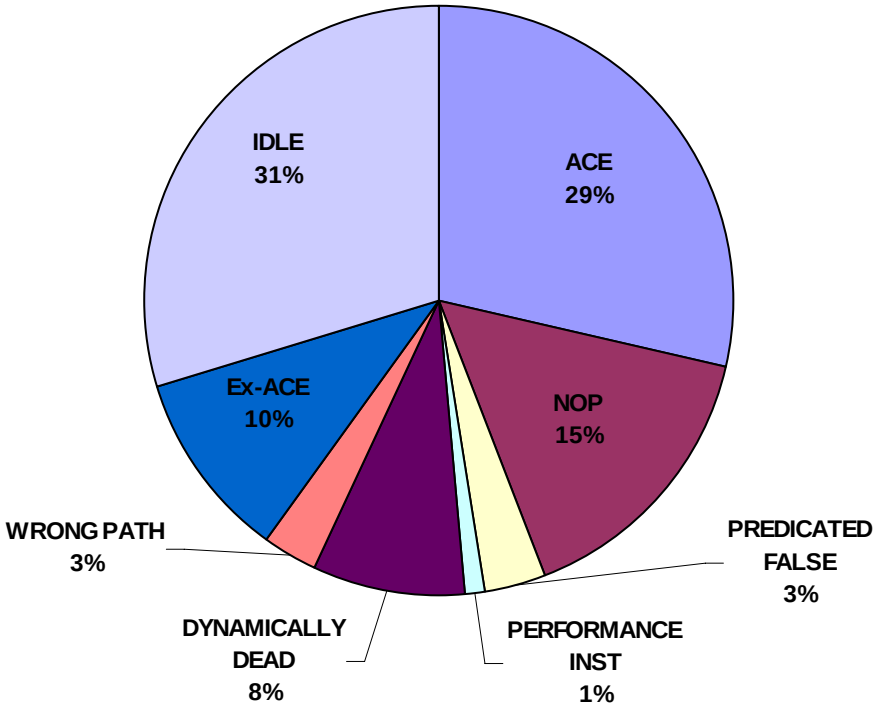
- 実行結果に影響しない構成要素
  - Functional units in idle
  - Predictors

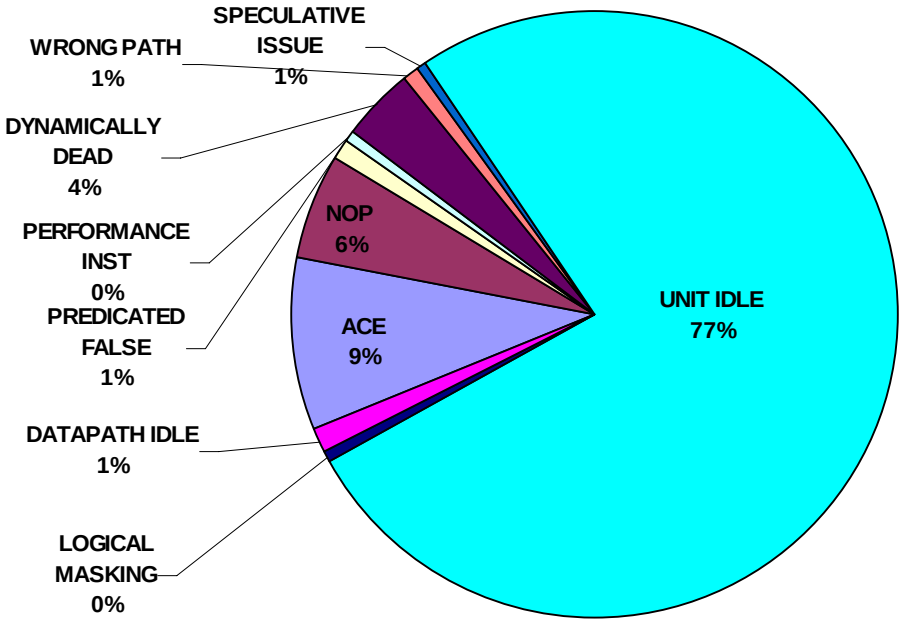
全てのソフトエラーがプログラム実行結果に影響するわけではない！

- AVF: Architectural Vulnerability Factor
  - プロセッサ内部で発生した故障がプログラム実行結果に現れる確率
- 例えば・・・
  - AVF = 0%
    - 分岐予測器
  - AVF = 100%
    - PC（プログラムカウンタ）
  - 0% < AVF < 100%
    - 命令キュー

AVFはどの程度か？

- ACE (Architecturally Correct Execution) bitの実装と計測
- Itanium-2 like Processor
- SPEC200





Instruction Queue
AVF = 29%

Functional Units
AVF = 9%

S. S. Mukherjee et al., "A Systematic Methodology to Compute the Architectural Vulnerability Factors of a High-Performance Microprocessor," Int'l Symp. On Microarchitecture, 2003.

「プログラムの実行」において信頼性を高めるには・・・

- MITF: Mean Instructions To Failure

$$\text{MITF} = \frac{\#\text{ instructions committed}}{\#\text{ errors encountered}}$$

$$= \frac{\text{IPC X (\# cycles)}}{\#\text{ errors encountered}}$$

$$= \frac{\text{IPC X Total time X frequency}}{\#\text{ errors encountered}}$$

$$= \text{IPC X MTTF X frequency}$$

$$= \frac{\text{IPC X frequency}}{\text{(Circuit Soft Error Rate) X AVF}}$$

$$= \frac{\text{IPC}}{\text{AVF}} \text{ X } \frac{\text{frequency}}{\text{Circuit Soft Error Rate}}$$

信頼性を高めるには性能を維持しつつAVFを小さくする事が重要！

$$\text{MITF} \propto \frac{\text{IPC}}{\text{AVF}}$$

AVF: Architectural Vulnerability Factor

Circuit Soft Error Rate：determined by alpha or neutron flux, circuit parameters, etc.

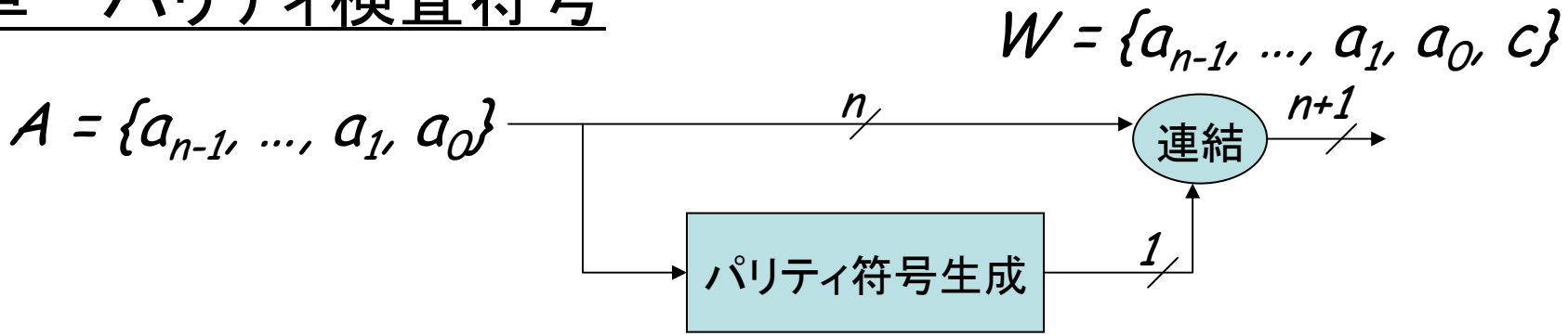
Christopher Weaver, Joel Emer, Shubhendu S. Mukherjee, Steven K. Reinhardt, "Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor," ISCA2004.

プロセッサの信頼性を高める！

	情報冗長化	時間冗長化	空間冗長化
Cache, Memory	•誤り検出/訂正符号		•多重化 •データの複製
Processor Core	•誤り検出/訂正符号	•命令再発行 •複製スレッド実行	•多重化

CacheやMemoryの信頼性を向上する（情報冗長化：単一ビット誤り検出）

単一パリティ検査符号



$c = a_{n-1} \oplus \ldots \oplus a_1 \oplus a_0$ （⊕は排他的論理和）

Wにおいて‘1’の数が偶数個（または奇数個）になるようc（パリティ検査ビット）を追加

例） A = “0010_1100” W = “0010_1100_1”

‘1’は偶数個！

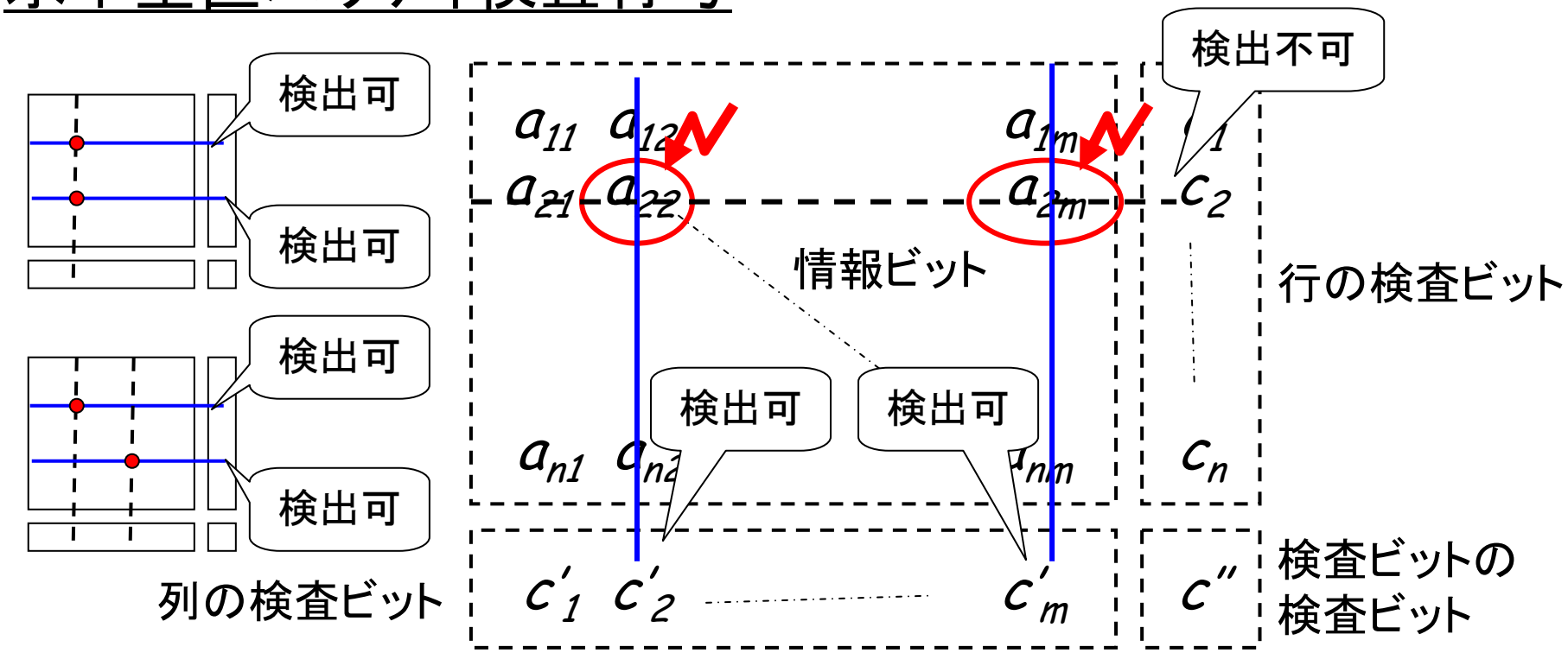
ソフトエラー発生！

W = “0000_1100_1”

‘1’は偶数個じゃない！

CacheやMemoryの信頼性を向上する（情報冗長化：二重ビット誤り検出）

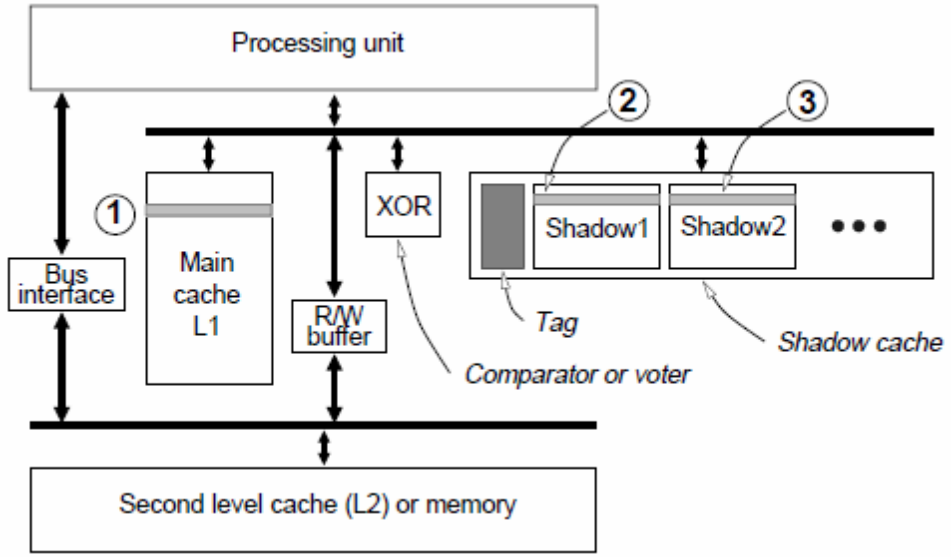
水平垂直パリティ検査符号



a_{11}～a_{nm}ビットをn×mの配列に並べ，行・列それぞれに検査ビットを追加する

CacheやMemoryの信頼性を向上する（空間冗長化：Shadow Caching ）

- 頻繁に参照されるキャッシュ・ブロックのみShadow Cacheに複製をコピー



S. Kim and A. Somani, "Area Efficient Architectures for Information Integrity in Cache Memories," ISCA'99

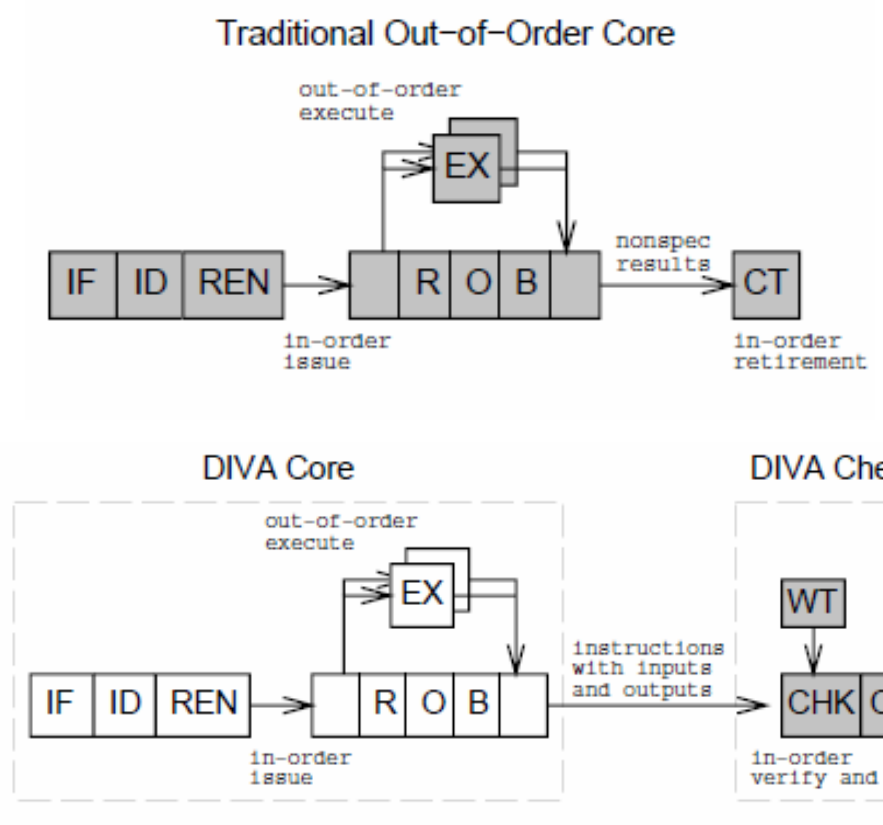
Processor Coreの信頼性を向上する
（時間/空間冗長化の基本アプローチ）
Replicate, Execute, and Compare

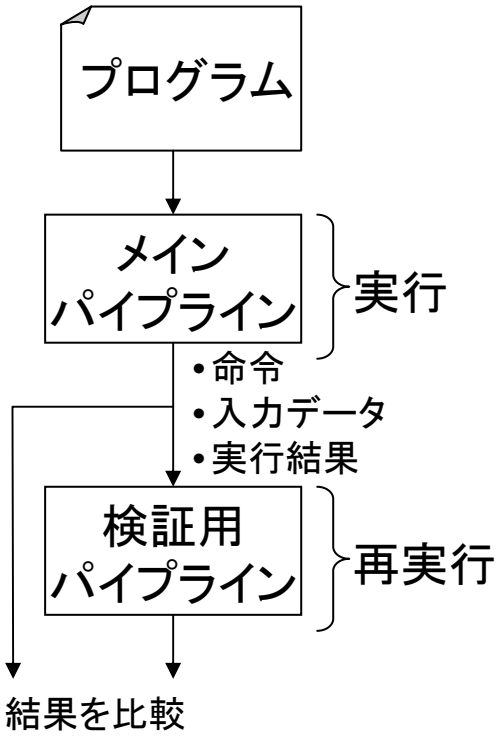
Replicate, Execute \ Compare		Cycle-by-Cycle	Instruction Commit
Spacial Redundancy (Hardware)	Processor Core	Himaraya	
	DataPath		DIVA, DIE
Temporal Redundancy (Program)	Thread		SRTR, AR-SMT
	Instruction		IRI

Himaraya: HP
DIVA: T. M. Austin, MICRO'99
DIE: J. Roy, MICRO'01

SRTR: T. N. Vijaykumar, ISCA'02
AR-SMT: E. Rotenberg, ISFTC'99
IRI: T. Sato, IEICE'03

DIVA
Dynamic Implementation Verification Architecture

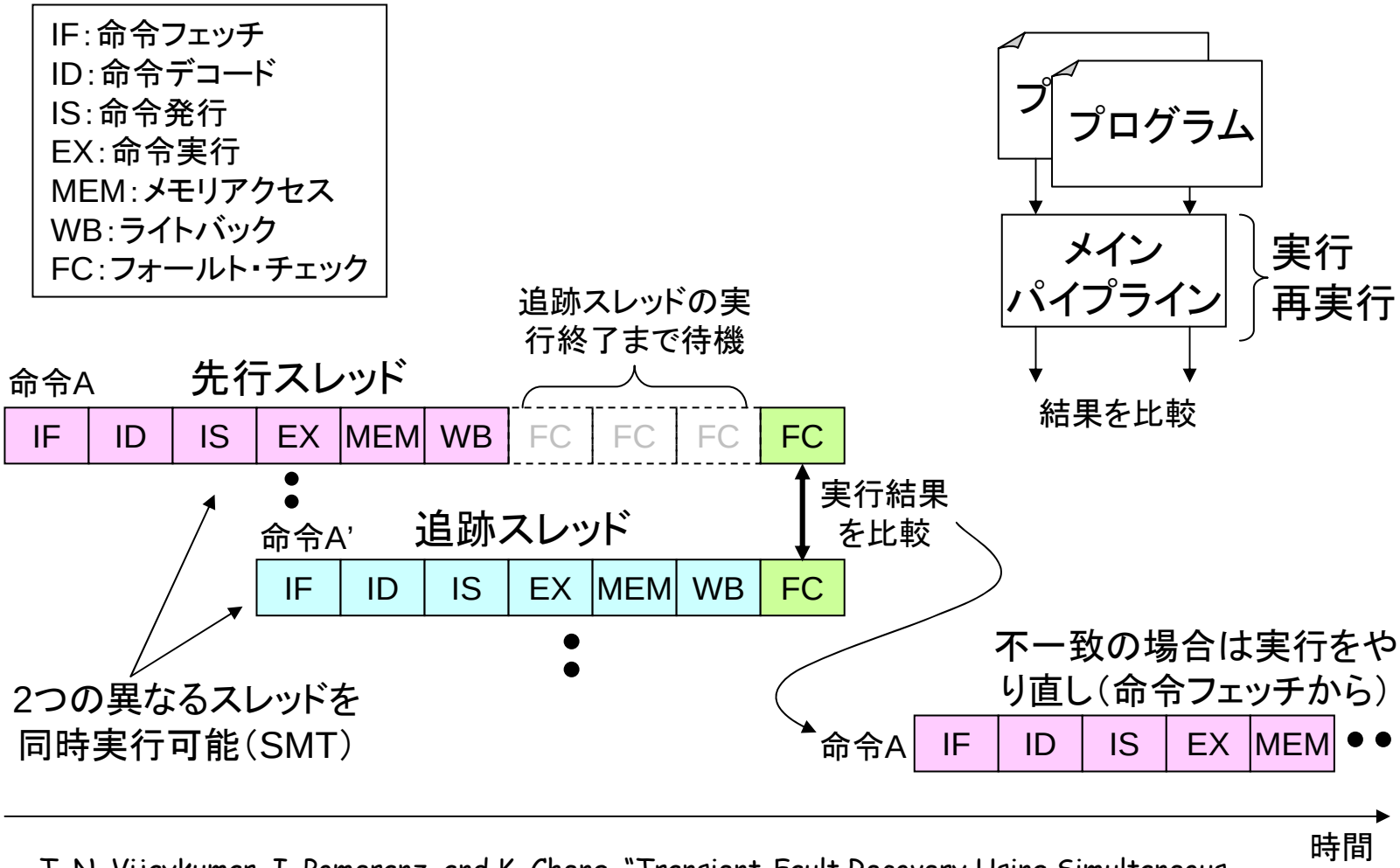




T. M. Austin, "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," International Symposium on Microarchitecture, pp.196-207, Dec. 1999.

SRTR

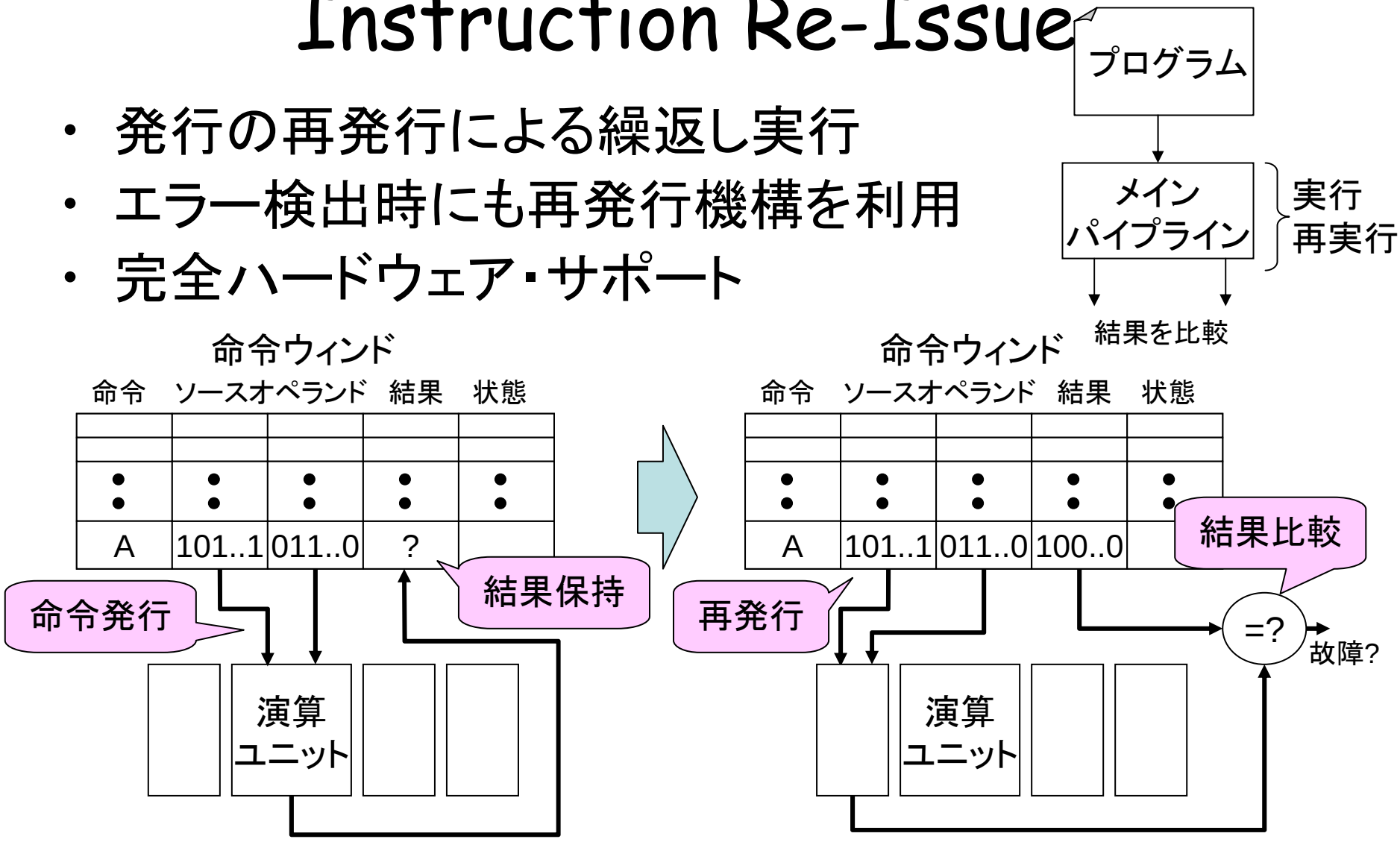
Simultaneously and Redundantly Threaded processors with Recovery



T. N. Vijaykumar, I. Pomeranz, and K. Cheng, "Transient-Fault Recovery Using Simultaneous Multithreading," International Symposium on Computer Architecture, pp.87-98, May 2002.

IRI
Instruction Re-Issue

- 発行の再発行による繰返し実行
- エラー検出時にも再発行機構を利用
- 完全ハードウェア・サポート



T. Sato, "A Transparent Transient Faults Tolerance Mechanism for Superscalar Processors," IEICE Transaction on Information and Systems, vol. E86-D, no. 12, Dec. 2003.

冗長実行による
性能/電力オーバヘッドを削減する

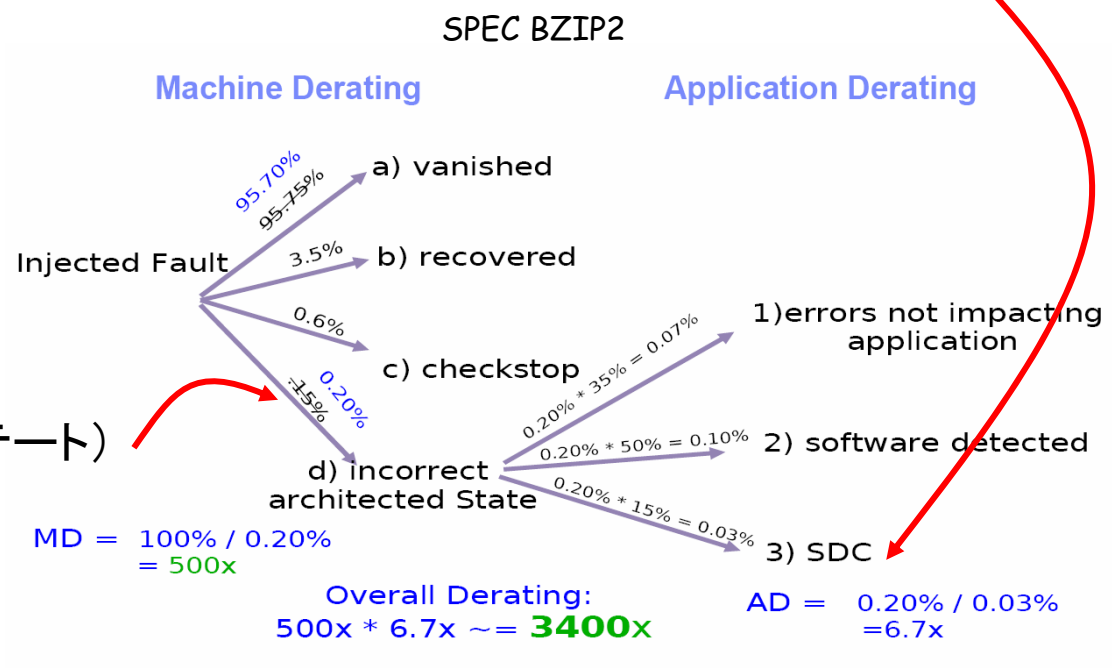
- 若干のカバレッジ低下を許す!
  - Partial Duplication(PD)[Gomma05]
  - Confident Prediction(CP)[Wang05]
  - PD+CP [Reddy06]
- 命令実行の特徴を利用する！
  - Narrow-Bit-Width [Sato04]
  - Self-Checking Instruction [Kumar06]

- M. A. Gomma and T. N. Vijaykumar, "Opportunistic Transient-Fault Detection," ISCA'05.
- N. J. Wang and S. J. Patel, "ReStore: Symptom based Soft Error Detection in Microprocessors," DSN'05.
- K. Sundaramoorthy, Z. Purser, and E. Rotenberg, "Slipstream Processors: Improving Both Performance and Fault Tolerance," ASPLOS'00.
- V. K. Reddy, S. Parhasarathy, and E. Rotenberg, "Understanding Prediction-Based Partial Redundant Threading for Low-Overhead, High-Coverage Fault Tolerance," ASPLOS'06.
- T. Sato, " Exploiting Sub-word Parallelism for Dependable Processors," *WSEAS Trans. on Information Science and Applications*, Vol. 1, No. 6, pp. 1051–1056, 2004.
- S. Kumar and A. Aggarwal, "Self-Cheking Instructions – Reducing Instruction Redundancy for Concurrent Error Detection," PACT'06.

POWER6 Microprocessor

実行結果にてSDCを発生させるのは
0.03%程度

エラー（誤ったアーキテクチャ・ステート）
を引き起こすのは0.2%程度



Jeffrey Kellington et. al., "*IBM POWER6 Processor Soft Error Tolerance Analysis Using Proton Irradiation* ," IEEE Workshop on Silicon Errors In Logic (SELSE3), Apr. 2007. (http://www.selse.org/selse07.program.linked.htm)

SPARC64V

Technology	90nm bulk CMOS, 10 layer Cu metallization
Chip size	18.46mm x 15.94mm
Clock Freq.	2.16GHz
Power	Max. 65W
CPU core	Single Core, 4 inst. decode OOO, ~240K Latches
Caches	L1$: 128KB+128KB, L2$:4MB

Defeating Factor for Latch Errors

		Freq.
Vanished Error		93.6%
Noticed Error	Recovered	4.9%
	Fatal Error	1.5%

	Error Detection	Recovery
L1 I$ Data, TLB	Parity	Invalid & Miss
BRHIS	Parity	Branch Miss-Pred. Recovery
L1I$ & L1D$ Tag	Parity + Duplication	Use correct data and rewrite
L1D$ Data, L2$ Data&Tag	SECDED	hardware ECC
Registers	Parity	Instruction Retry
ALU, Shifter, VIS	Parity Prediction	
Mult/Div	Residlue check + Parity Prediction	

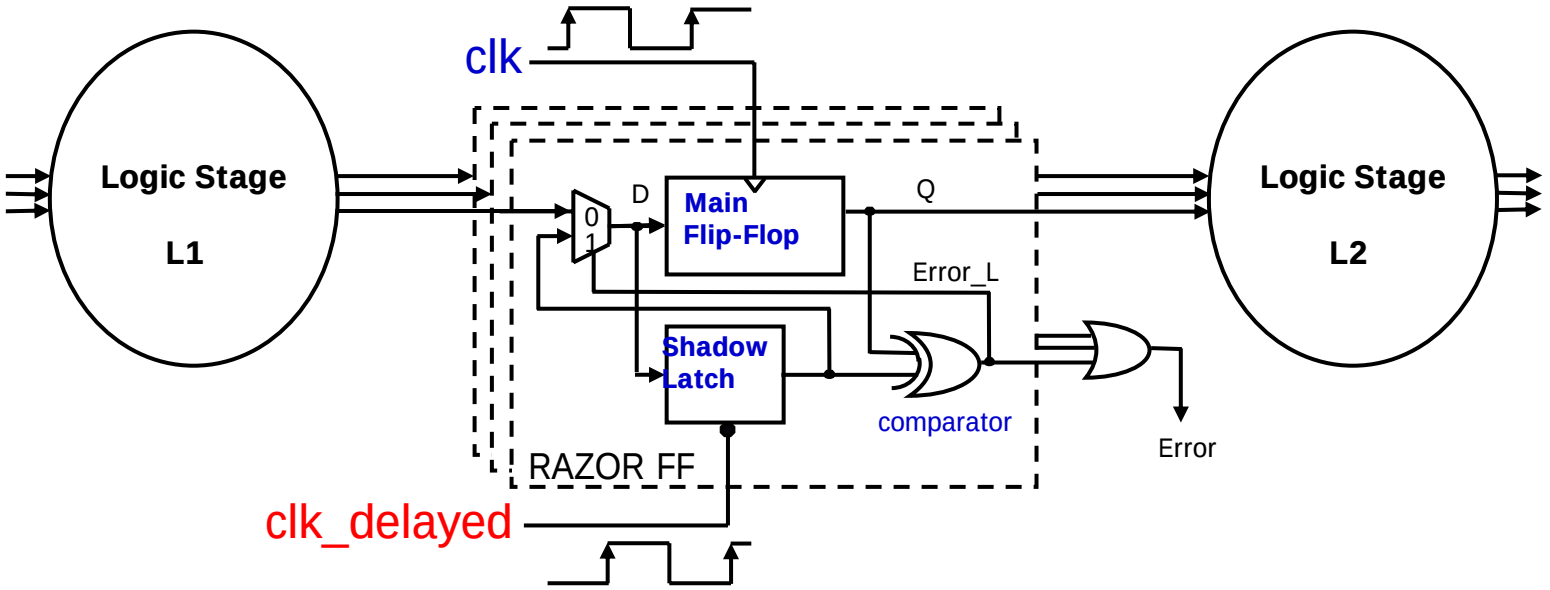
Jeffrey Kellington et. al., "*IBM POWER6 Processor Soft Error Tolerance Analysis Using Proton Irradiation* ," IEEE Workshop on Silicon Errors In Logic (SELSE3), Apr. 2007. (http://www.selse.org/selse07.program.linked.htm)

チュートリアル内容

- 信頼性（耐故障性）を向上する！
  - ソフトエラー対策
  - タイミングエラー対策
- 安全性を向上する！
  - 不正プログラムの実行防止
  - メモリデータ改ざんの防止
  - 情報漏えいの防止
- プロセスばらつきの影響を緩和する！
  - 製造後のパイプライン・チューニング
- ハードウェアバグを回避する！
  - ハードウェア・パッチング
- まとめ

Razor

- 2つのクロック信号
  - clk：最悪ケースを保障しない, clk_delayed：最悪ケースを保障する
  - エラーの検出⇒2つの出力を比較



Dan Ernst et. al., "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," Proc. of the 36th International Symposium on Microarchitecture, 2003.

SACSIS2007でも・・・

- 5月25日(最終日) 11:00～12:30 高信頼化
  - 「カナリア・フリップフロップを利用する省電力マイクロプロセッサの評価」 佐藤 寿倫（九大）
    （最優秀論文）
  - 「レジスタファイルの書き込み時タイミングエラーの検出・回復手法」 入江 英嗣（JST），杉本 健，五島 正裕，坂井 修一（東大）
- こちらにご参加下さい！

信頼性を向上する(まとめ)

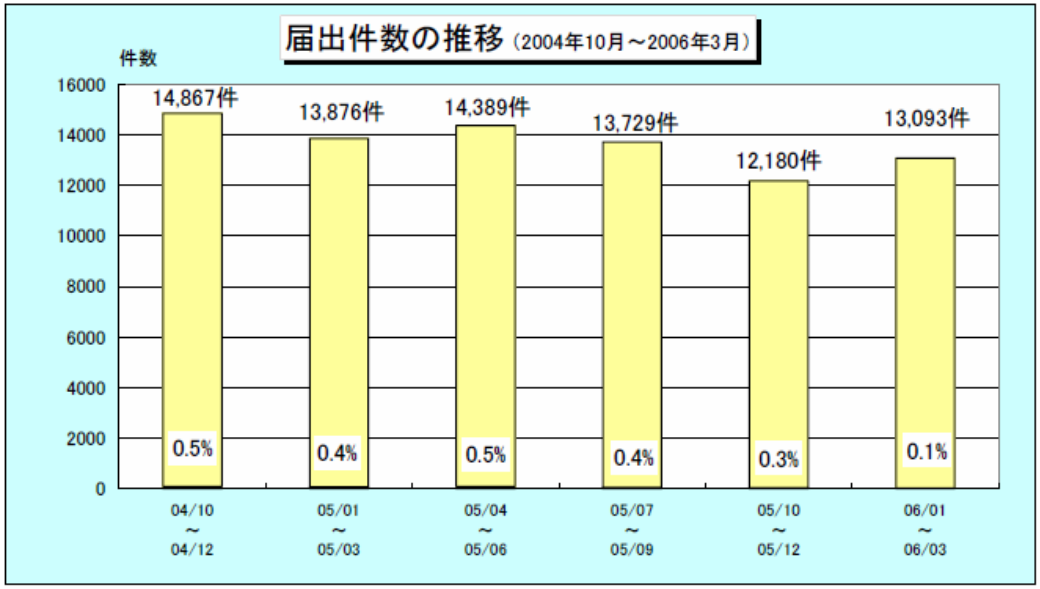
- ポイント
  - 如何に効率よく冗長化するか(どれだけさぼるか)?
  - 全ての「故障」がエラーを引き起こす訳ではない!
    - メモリやラッチに関してはECCやパリティが強力
    - ハイエンド・プロセッサ(0.2～1.5%)
    - ローエンド・プロセッサでは?
- 今後の課題と展望
  - 消費電力/エネルギーとのトレードオフ
    - 面積/性能オーバヘッドの努力は依然として必要
  - ロジックでのソフトエラーの影響は?
  - 様々な応用展開
    - 高性能化，低消費電力/エネルギー化，など

チュートリアル内容

- 信頼性(耐故障性)を向上する！
  - ソフトエラー対策
  - タイミングエラー対策
- 安全性を向上する！
  - 不正プログラムの実行防止
  - メモリデータ改ざんの防止
  - 情報漏えいの防止
- プロセスばらつきの影響を緩和する！
  - 製造後のパイプライン・チューニング
- ハードウェアバグを回避する！
  - ハードウェア・パッチング
- まとめ

コンピュータ・ウィルス問題（どの程度の被害があるのか？）

- 国内では2ヶ月で12,000件以上の届出（IPA）
- 被害総額（推計）
  - 国内で3,025億円，米国で130億ドル

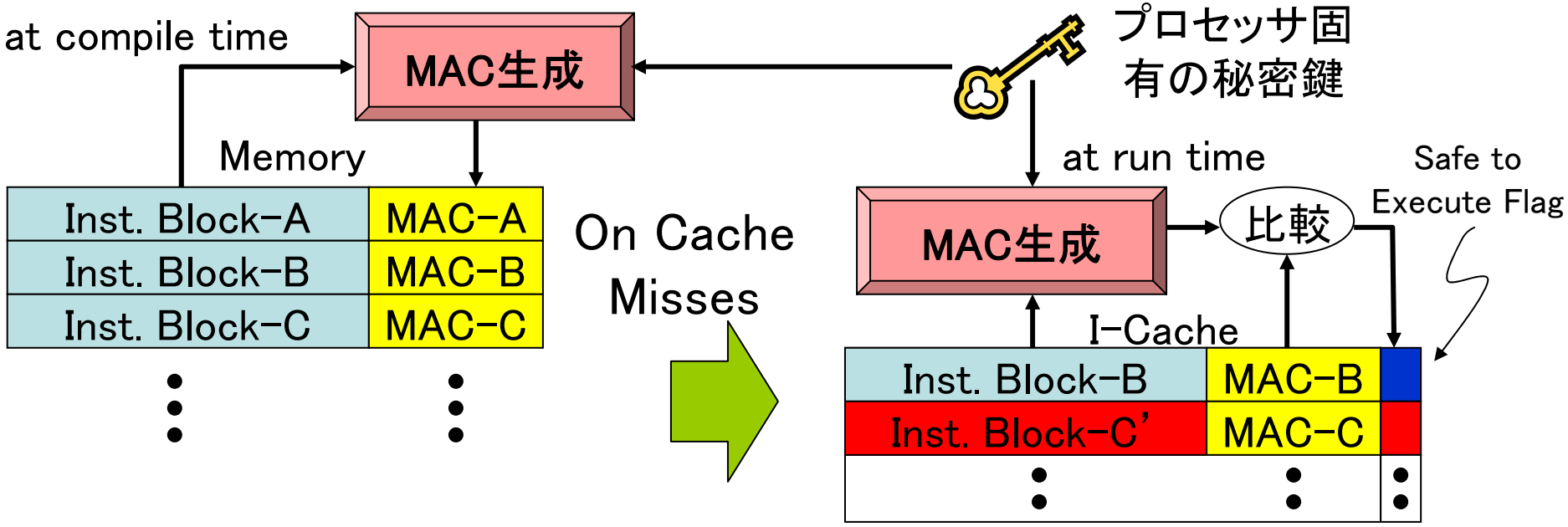


IPA2006 年第1 四半期[1 月～3 月]コンピュータウイルス届出状況より
http://www.ipa.go.jp/security/txt/2006/documents/2006q1-v.pdf

IPA国内・海外におけるコンピュータウイルス被害状況調査　被害額推計 報告書(2004/4)より
http://www.ipa.go.jp/security/fy15/reports/virus-survey/documents/2003_calc_model.pdf

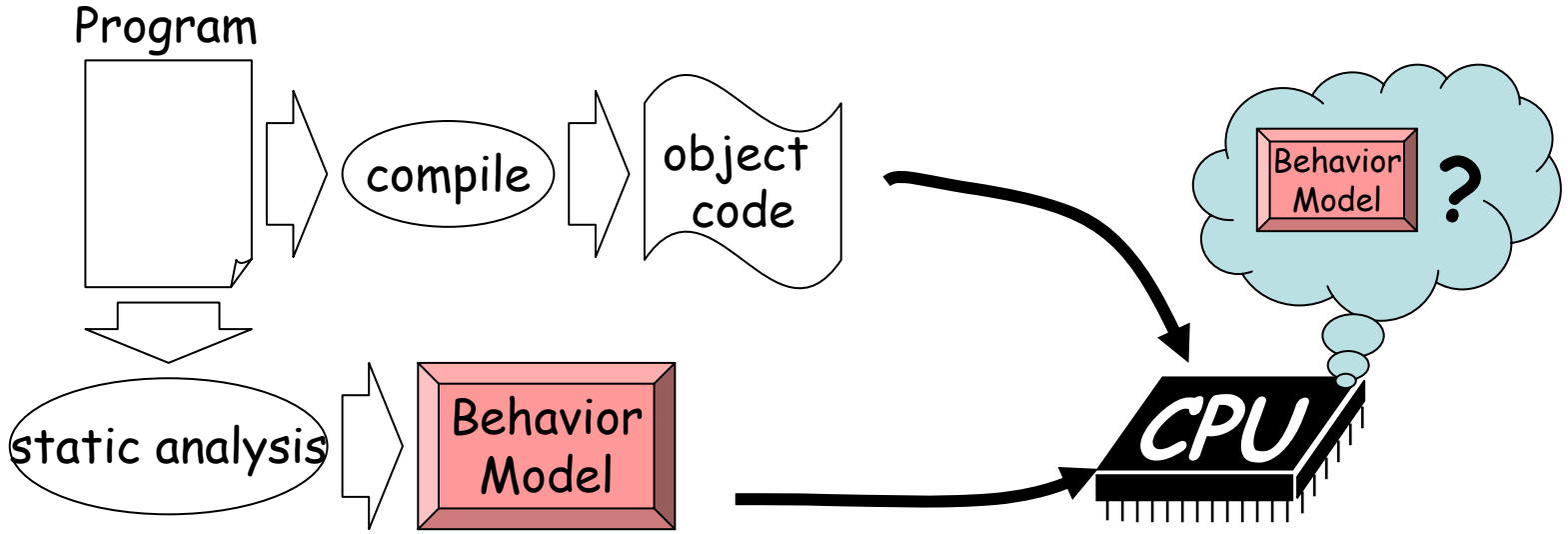
メッセージ認証コード
（MAC: Message Authentication Code）

- 任意長の平文と鍵情報から生成される「固定長のメッセージ」
  - CBC-MACやHMAC（ハッシュ関数の利用）など
- 命令ブロック毎にMACを計算し追加
  - メモリ領域の拡張が必要
- 命令キャッシュへのフィル時にMACをチェック
  - MACを改ざんされてもプロセッサ固有鍵の秘密性が保たれていれば検出可能



実行振舞いモデル(振舞い抽出方式)

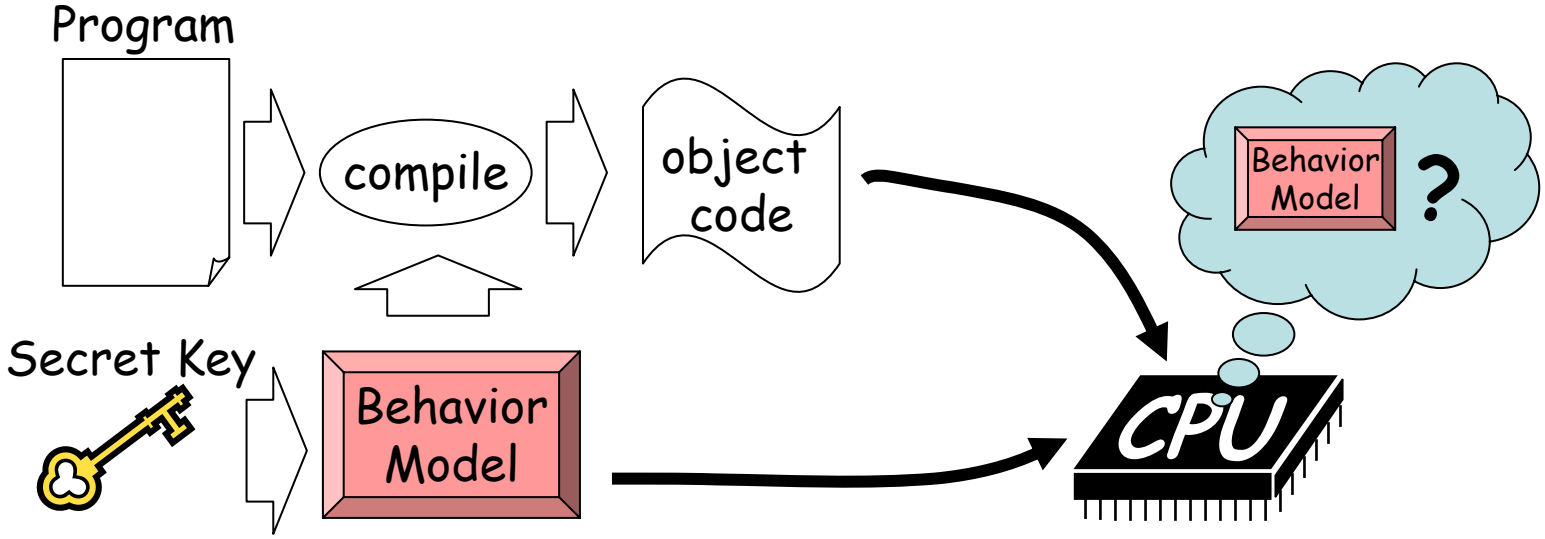
- プログラムコードからその実行振舞いをモデル化
  - 使用するシステム・コールなど
- プログラム実行時に「振舞いモデル」と同じ動作をするか監視
- 振舞いモデルを満足する不正プログラムは検出不可



David Wagner and Drew Dean, "Intrusion Detection via Static Analysis," IEEE Symp. on Security and Privacy (S&P), 2001

実行振舞いモデル（振舞い制御方式）

- 秘密鍵よりプログラム認証のための「実行の振舞い」を決定
  - メモリアクセス・パタンなど
- 決定した実行振舞いを再現するようコード生成
- 実行時に「実行の振舞い」を監視

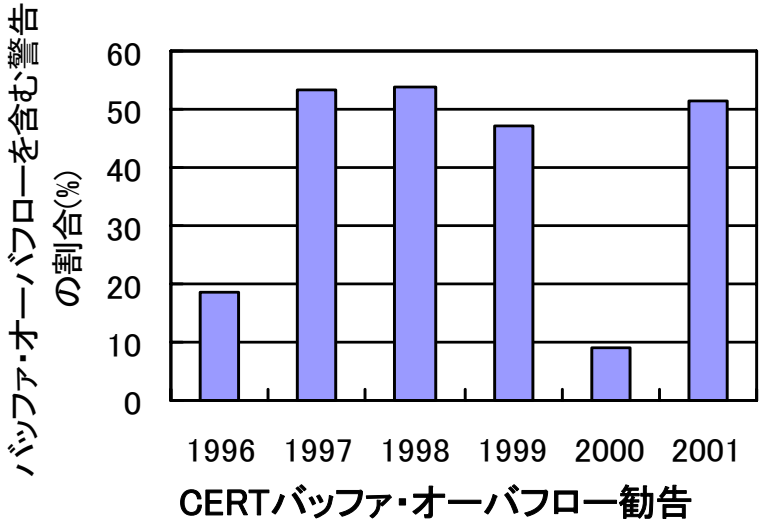


T. Iwasa and K. Inoue, "FPGA Implementation of a Secure Microprocessor," Workshop on Architecture Research using FPGA Platforms, Feb. 2005.

バッファ・オーバフロー攻撃

- 多く活用される脆弱性の1つ
  - Blaster@2003,
  - CodeRed@2001

- メカニズム
  - データ境界を越えた書込み
    - C標準ライブラリ内にも存在
    - strcpyなど
  - スタックの破壊（スタック・スマッシング）
    - 攻撃コードの挿入と戻りアドレスの改ざん
  - プログラム実行制御の乗っ取り
    - 改ざんされた戻りアドレスがPCに設定



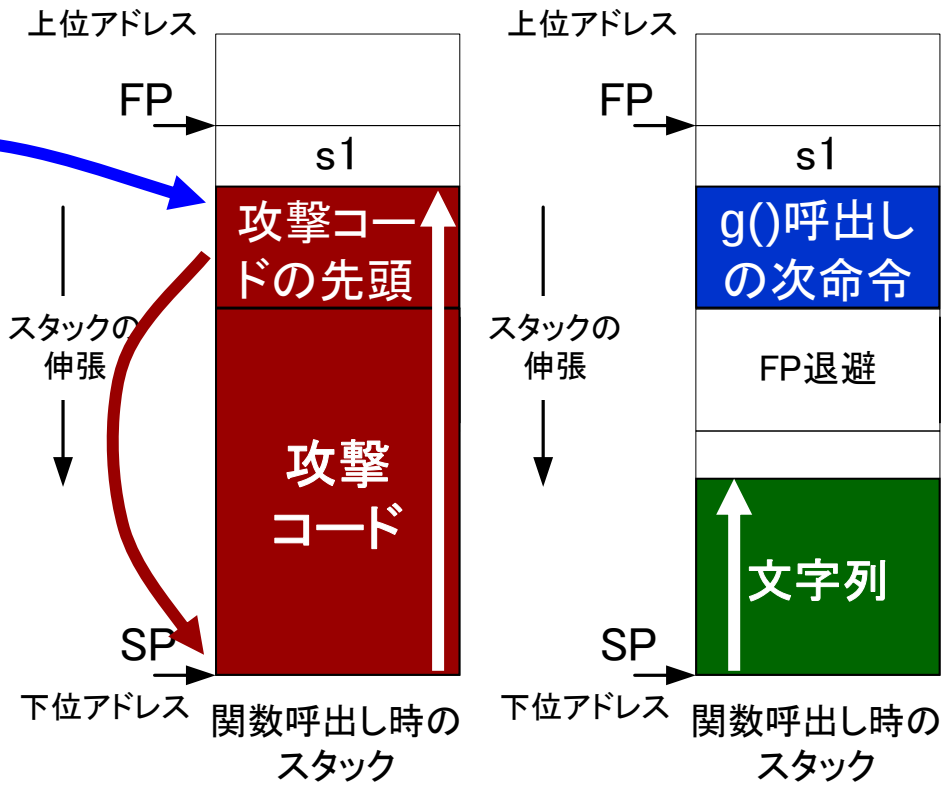
R.B.Lee, D.K.Karig, J.P.McGregor, and Z.Shi, “Enlisting Hardware Architecture to Thwart Malicious Code Injection,” Proc. of the Int. Conf. on Security in Pervasive Computing, Mar. 2003.

スタック・スマッシングによる実行制御の乗っ取り

実行コード

```
int f ( ) {
  …
  g (s1);
  …
}

int g ( char *s1) {
  char buf [10];
  …
  strcpy(buf, s1);
  …
}
```

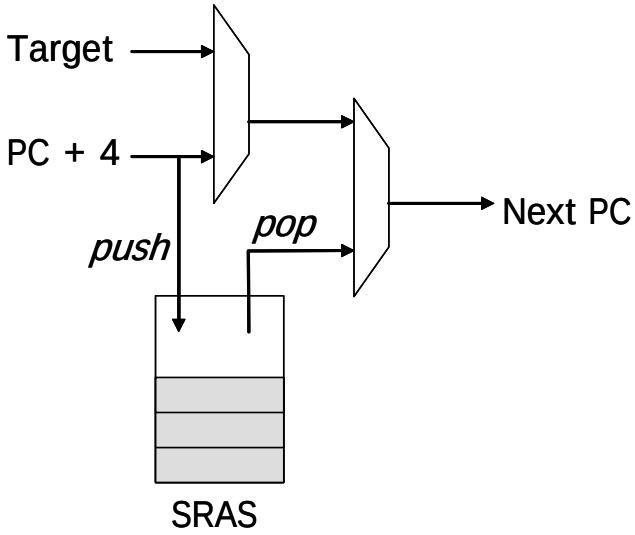


処理手順

1. 関数f ( )の実行
2. 関数g( )の呼出し
3. 文字列コピー
4. 関数f( )へ復帰

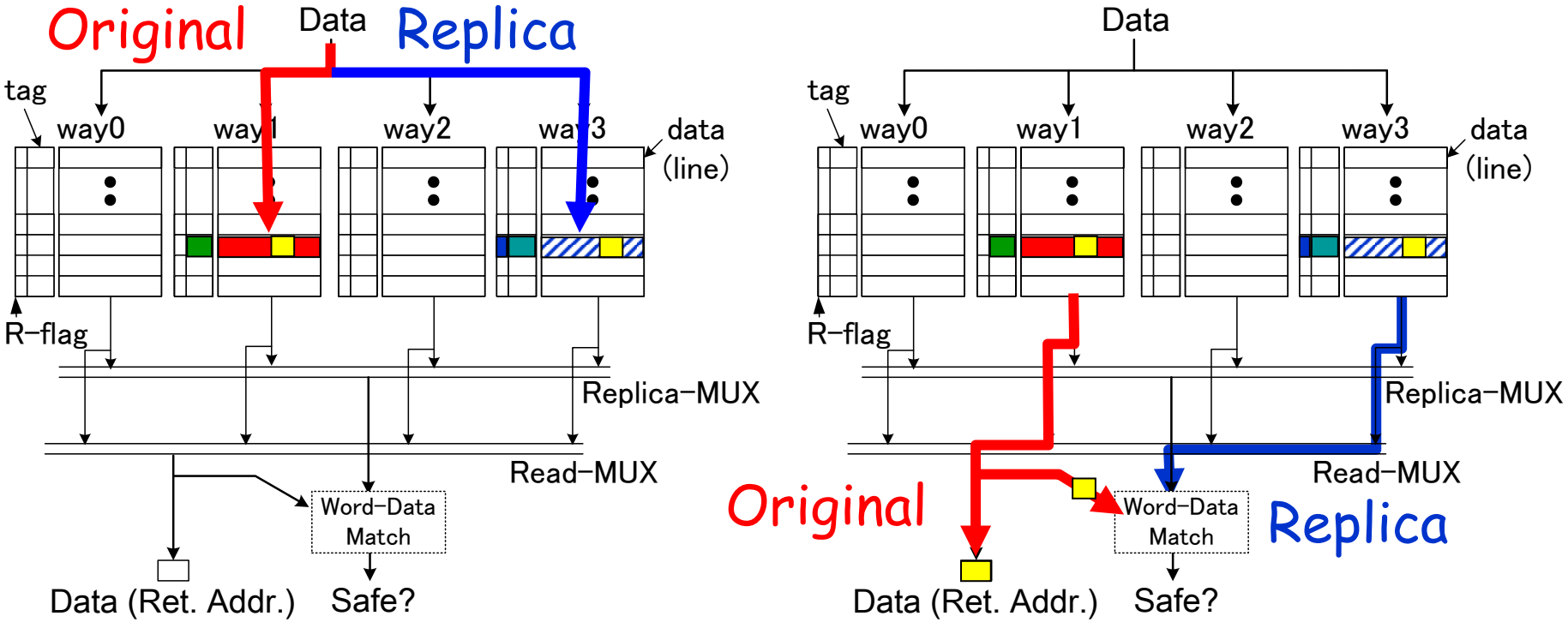
バッファオーバフローの動的検出
～SRAS：Secure Return Address Stack～

- ハードウェアによる戻りアドレス改ざんの検出
  - 戻りアドレス値のコピーをプロセッサ内部の安全なメモリ領域に保存
  - LIFOアクセスにのみ対応
- 動作
  - call命令発行時の戻りアドレスをCPU内部の専用スタック(SRAS)に格納
  - return命令発行時にSRASから戻り番地を入手
  - メモリ・スタックからポップと比較
  - 不一致なら改竄発生！



バッファオーバフローの動的検出
～Secure Cache～

- ハードウェアによる戻りアドレス改ざんの検出
  - 戻りアドレス書込み時に複製（レプリカ・ライン）を作成
  - 戻りアドレス読出し時に複製と比較
  - LIFO以外のアクセスにも対応



K. Inoue, ``Energy-Security Tradeoff in a Secure Cache Architecture Against Buffer Overflow Attacks,'' ACM Computer Architecture News, vol33, no.1, pp.81-89, Mar. 2005.

入力データ追跡による実行乗っ取りの検出

- 多くの場合，「悪意あるデータの入力」によりプログラムの実行制御を乗っ取る

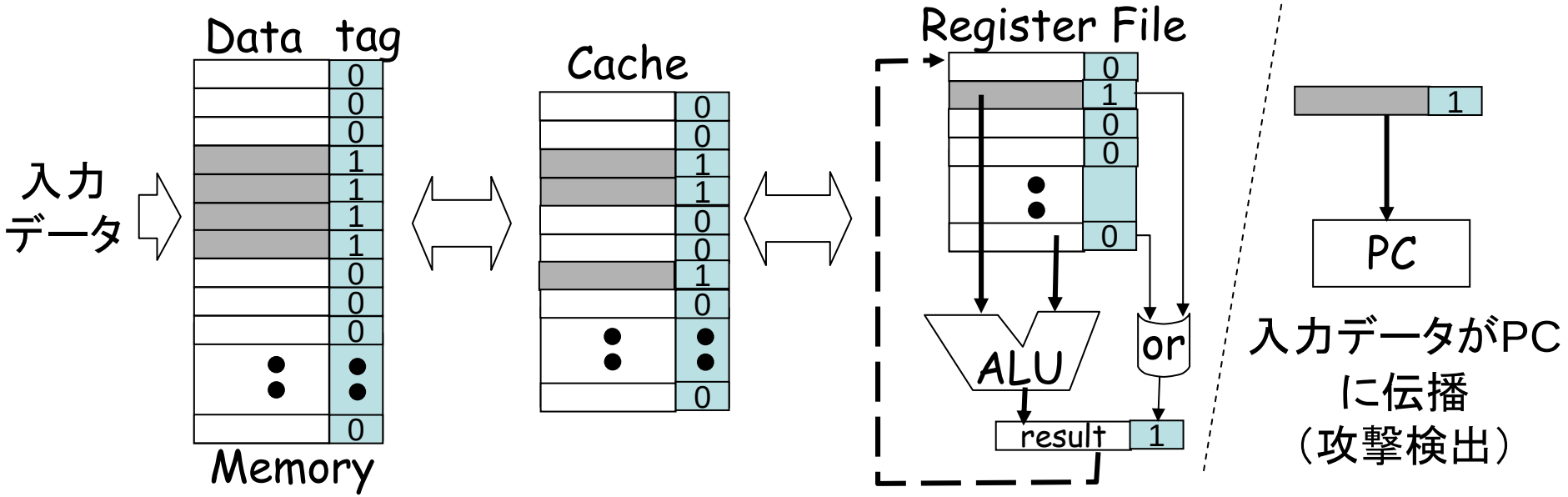
脆弱性のあるプログラムコード例
（バッファオーバフロー攻撃の対象）

```
void foo ( ) {
  char buf[10];
  scanf(“%s”, buf);
}
```

- アプローチ
  - 外部入力データの利用状況をプログラム実行時に追跡
    - メモリデータを「データの値＋タグ（外部入力データか否かを示す）」の組で記憶
    - 実行時に外部入力データの伝播を追跡
  - 予め定めたセキュリティ・ポリシに基づき攻撃を検出
    - （例）外部入力データがPC（プログラム・カウンタ）の値に影響を与える場合には何らかの攻撃が発生したと判定
    - など

入力データの追跡

- アーキテクチャ・サポート
  - 主記憶，キャッシュ，レジスタ・ファイル等のデータ記憶領域に対してエントリ毎にタグビットを追加
  - 演算実行時のタグビット伝播機能
  - 入力データ（タグビットがセットされたデータ）の利用モニタリング機能
- OSサポート
  - 入力データをメモリに格納する際にタグビットを“1”にセット



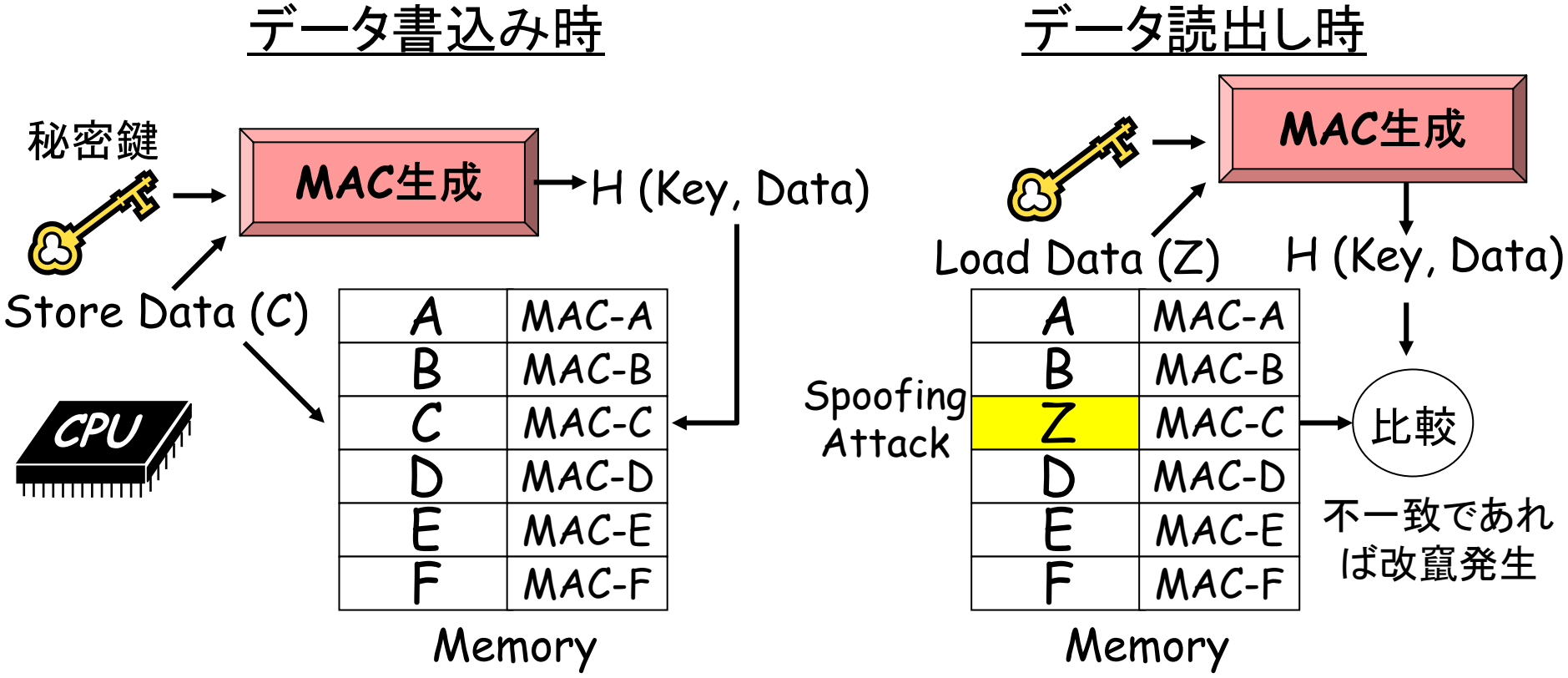
勝沼他，“アドレスオフセットに着目したデータフロー追跡による注入攻撃の検出，” SACSIS, 2006.
S. Chen et. Al, “Defending Memory Corruption Attacks via Pointer Taintedness Detection,” DSN, 2005.

チュートリアル内容

- 信頼性(耐故障性)を向上する！
  - ソフトエラー対策
  - タイミングエラー対策
- 安全性を向上する！
  - 不正プログラムの実行防止
  - メモリデータ改ざんの防止
  - 情報漏えいの防止
- プロセスばらつきの影響を緩和する！
  - 製造後のパイプライン・チューニング
- ハードウェアバグを回避する！
  - ハードウェア・パッチング
- まとめ

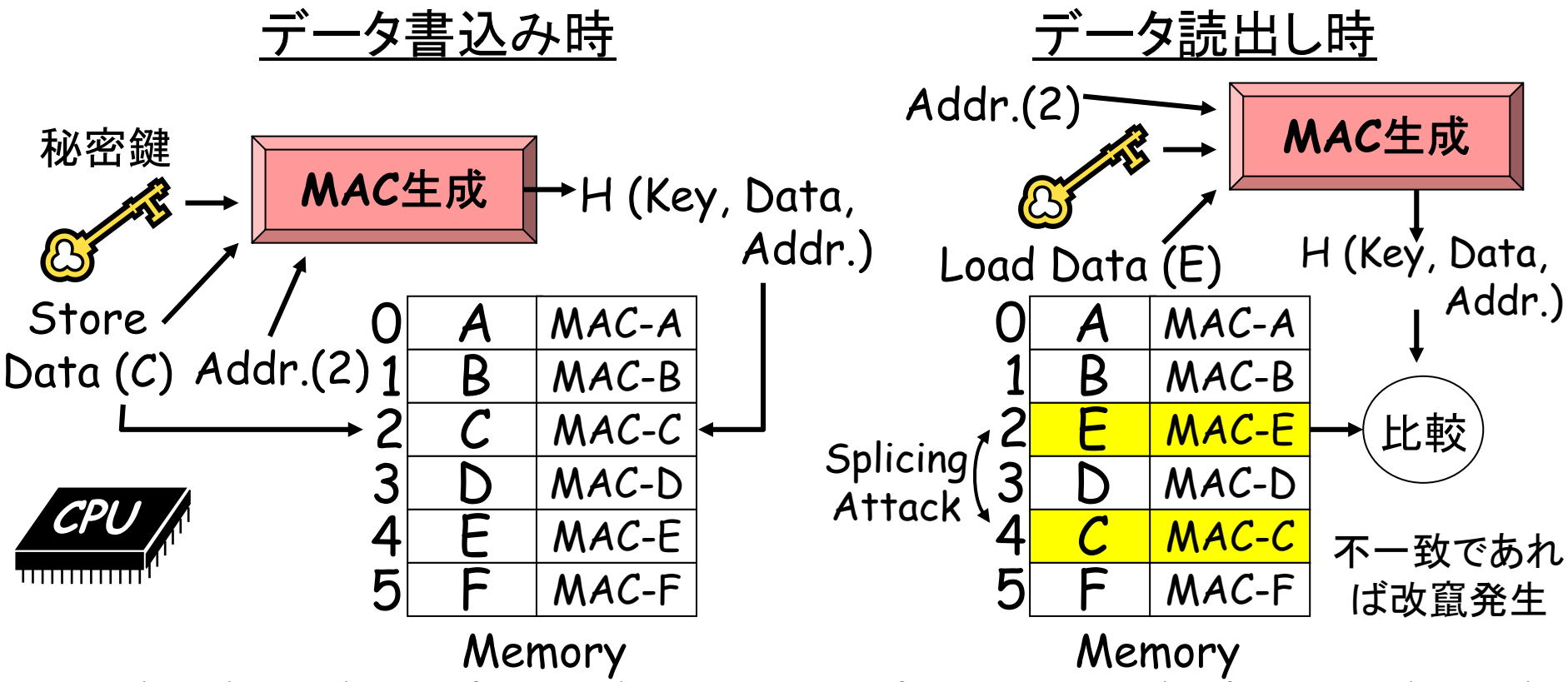
メッセージ認証コードの利用
～Spoofing Attackを検出する～

- 「誰が」書き込んだデータかを記憶
  - MAC（Message Authentication Code）を利用
  - 「書込みデータ＋秘密鍵」でMACを生成し，「データとMAC」をメモリに記憶



メッセージ認証コードの利用
～Splicing Attackも検出する～

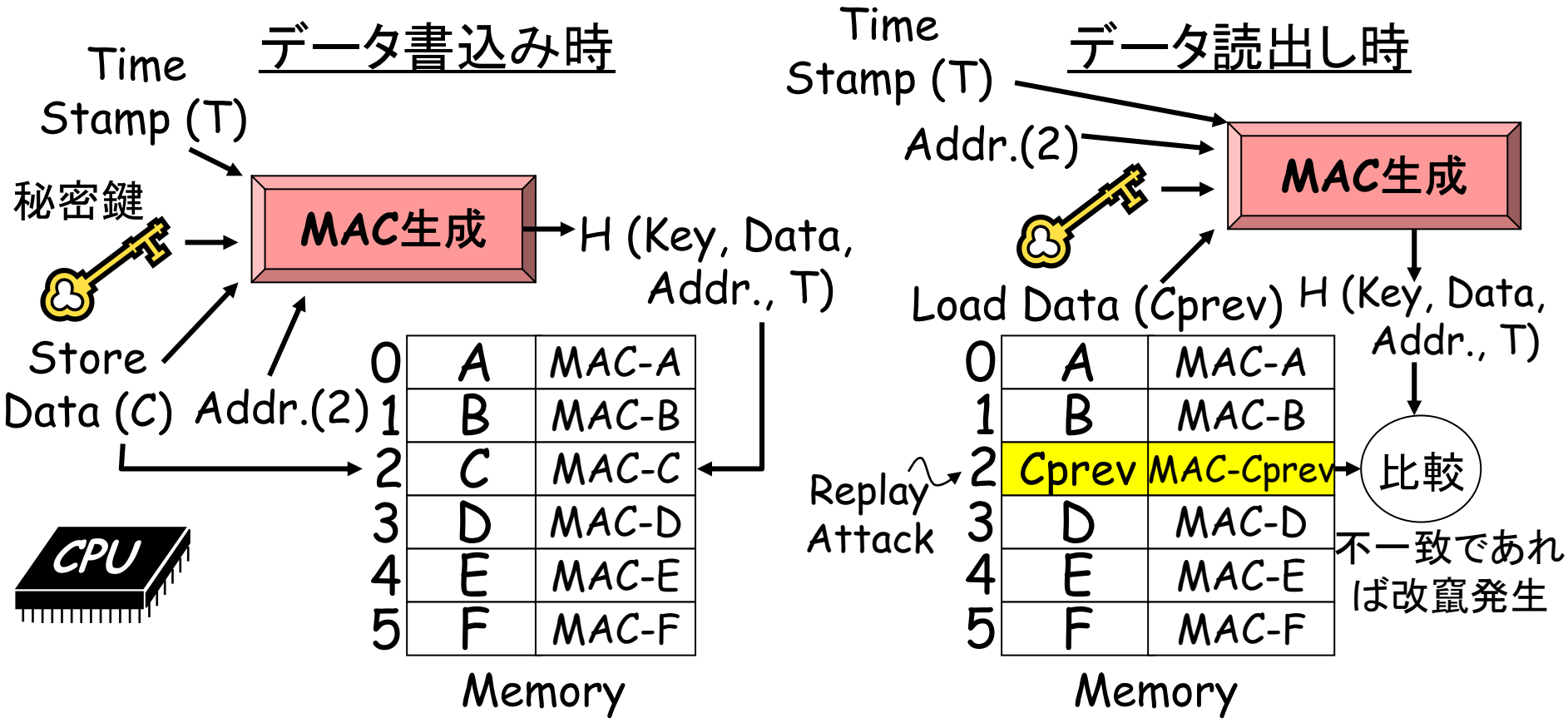
- 「誰が，どこに」書込んだデータかを記録
  - 「書込みアドレス＋書込みデータ＋秘密鍵」でMACを生成



D. Lie et. al., "Architectural Support for Copy and Tamper Resistant Software," International Conference on Architectural Support for Programming Languages and Operating Systems, pp.168-177, Dec. 2000.

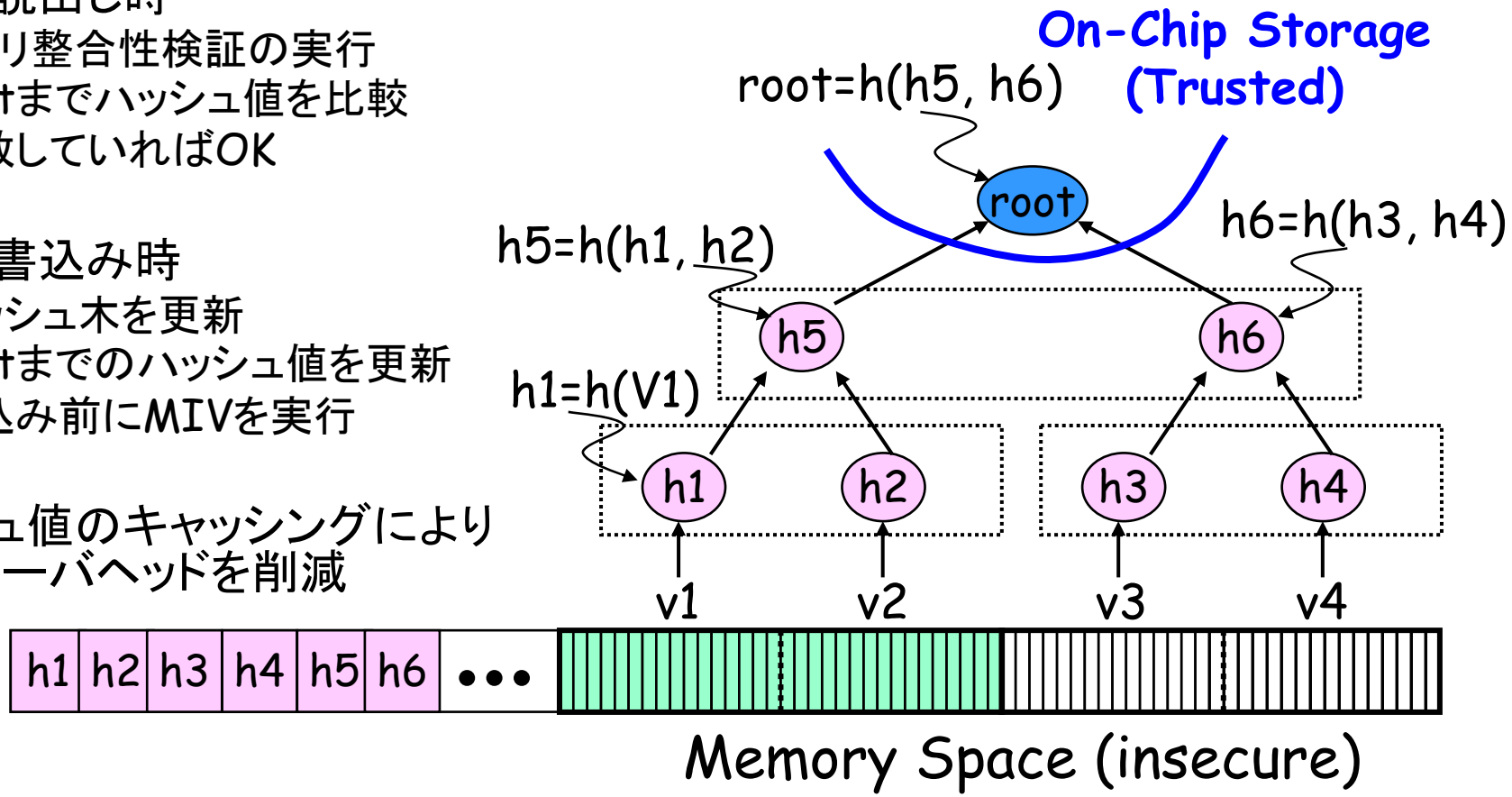
メッセージ認証コードの利用
～Replay Attackも検出する～

- 「いつ，誰が，どこの」書込んだデータかを記録
  - 「書込み時刻＋書込みアドレス＋書込みデータ＋秘密鍵」でMACを生成



ハッシュ木によりメモリ・イメージのダイジェストを保存する！

- データ読出し時
  - メモリ整合性検証の実行
  - rootまでハッシュ値を比較
  - 一致していればOK
- データ書込み時
  - ハッシュ木を更新
  - rootまでのハッシュ値を更新
  - 書込み前にMIVを実行
- ハッシュ値のキャッシングにより性能オーバヘッドを削減



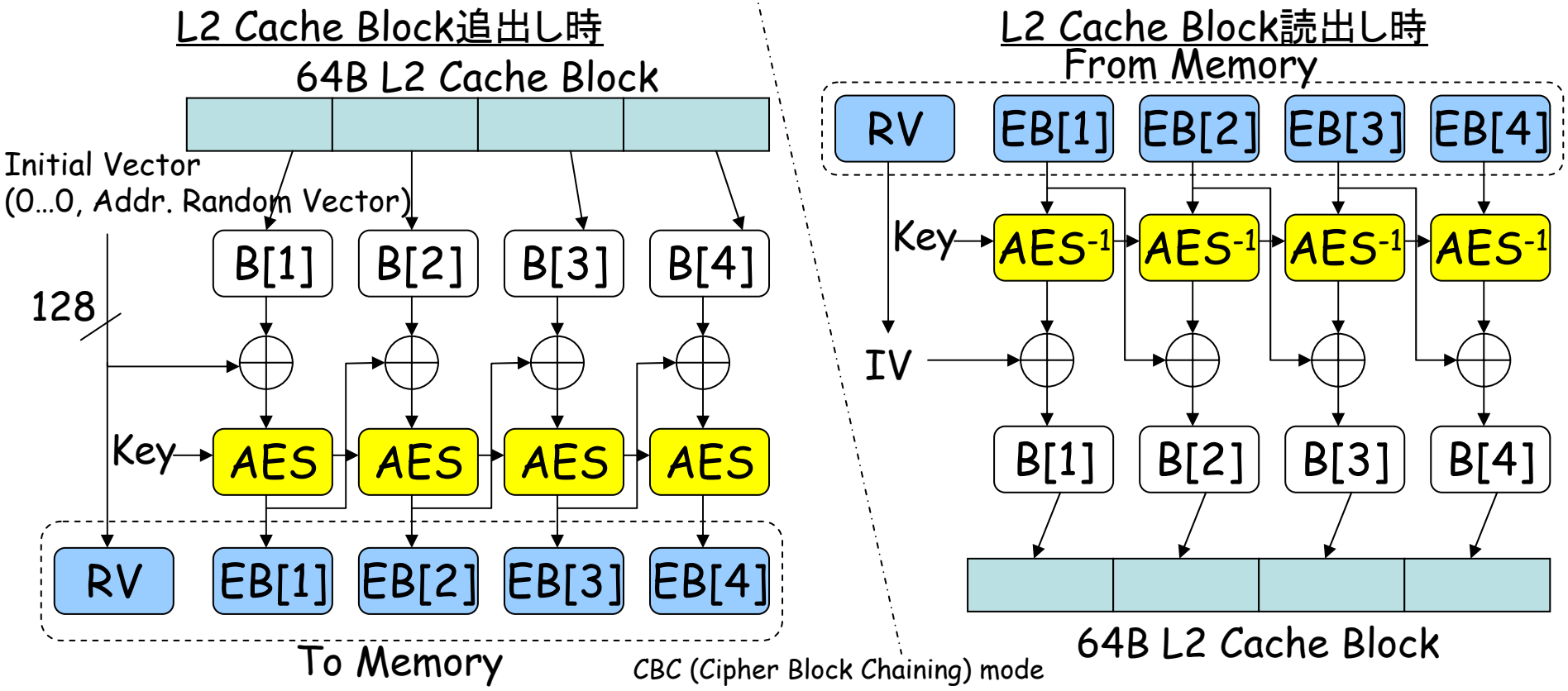
G. E. Suh, D. Clarke, B. Gassend, M. Dijk, and S. Devadas, "Efficient Memory Integrity Verification and Encryption for Secure Processors," International Symposium on Microarchitecture, pp.339-350, Dec. 2003.

チュートリアル内容

- 信頼性(耐故障性)を向上する！
  - ソフトエラー対策
  - タイミングエラー対策
- 安全性を向上する！
  - 不正プログラムの実行防止
  - メモリデータ改ざんの防止
  - 情報漏えいの防止
- プロセスばらつきの影響を緩和する！
  - 製造後のパイプライン・チューニング
- ハードウェアバグを回避する！
  - ハードウェア・パッチング
- まとめ

メモリデータの情報漏洩を防止する！（Direct Block Encryption）

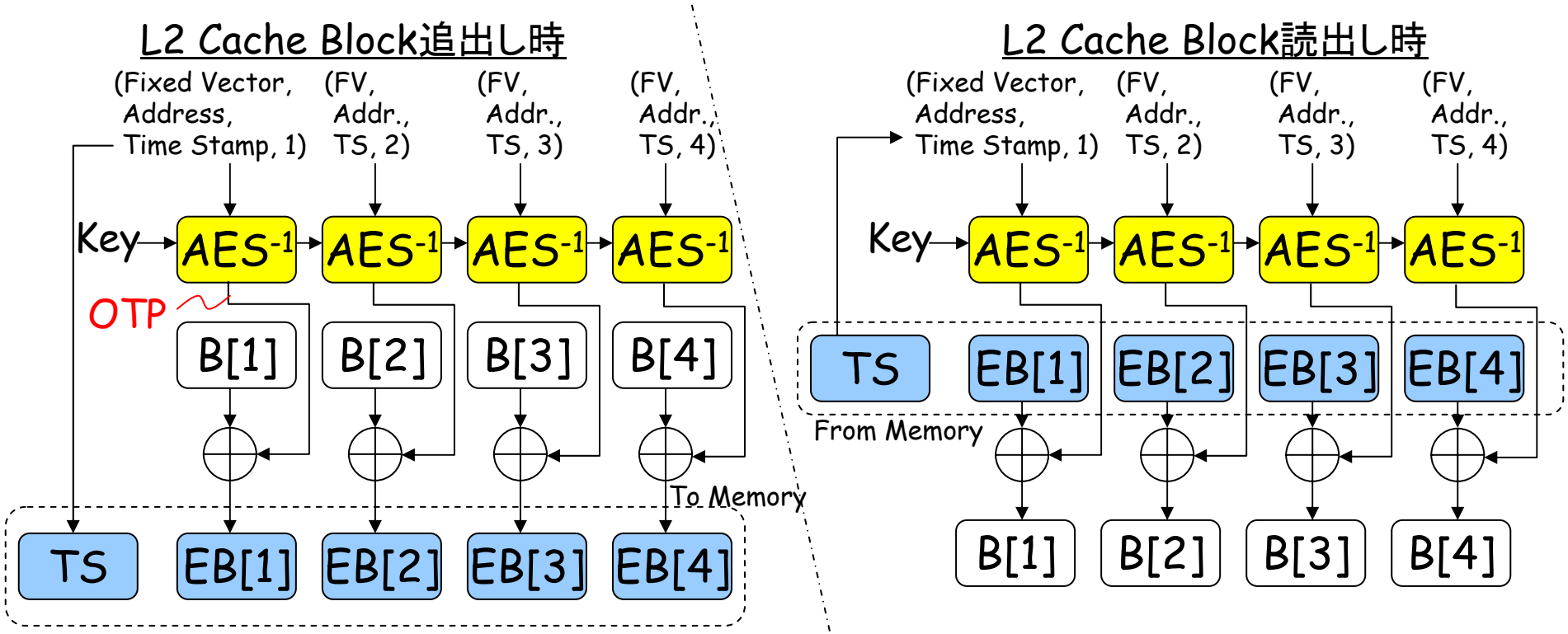
- オフチップ・メモリに格納するデータを全て暗号化
- AES（Advanced Encryption Standard）など
- キャッシュ・ミス時に暗号/復号処理



D. Lie et. al., "Architectural Support for Copy and Tamper Resistant Software," International Conference on Architectural Support for Programming Languages and Operating Systems, pp.168-177, Dec. 2000.

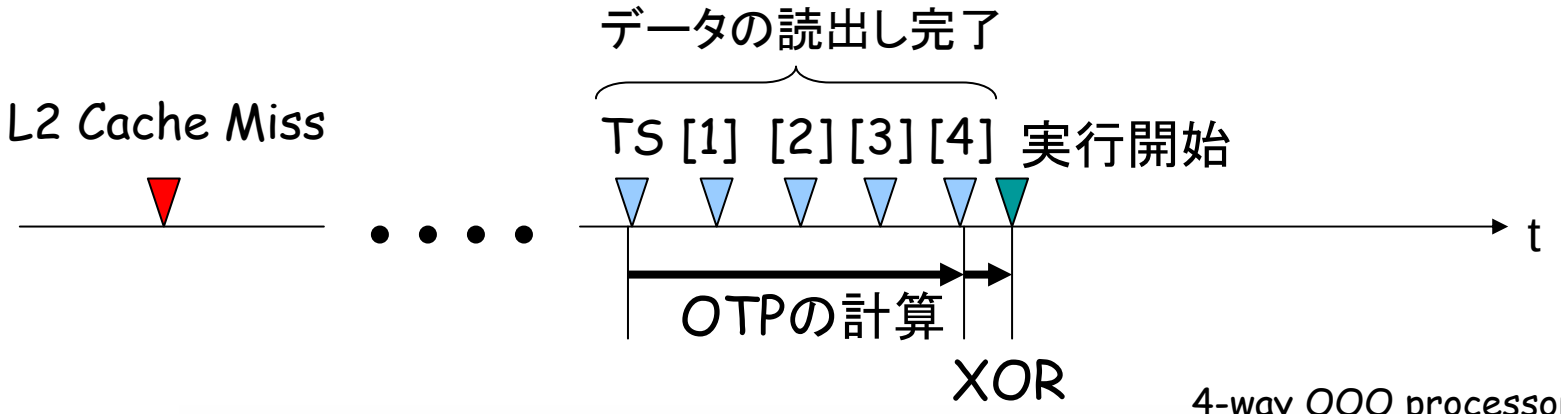
性能オーバヘッドの低減
OTP (One-Time Pad)

- AESを用いて間接的に記憶データを暗号化
- 記憶データに依存しないOTPを生成
- TS（Time Stamp）をデータと一緒にストア

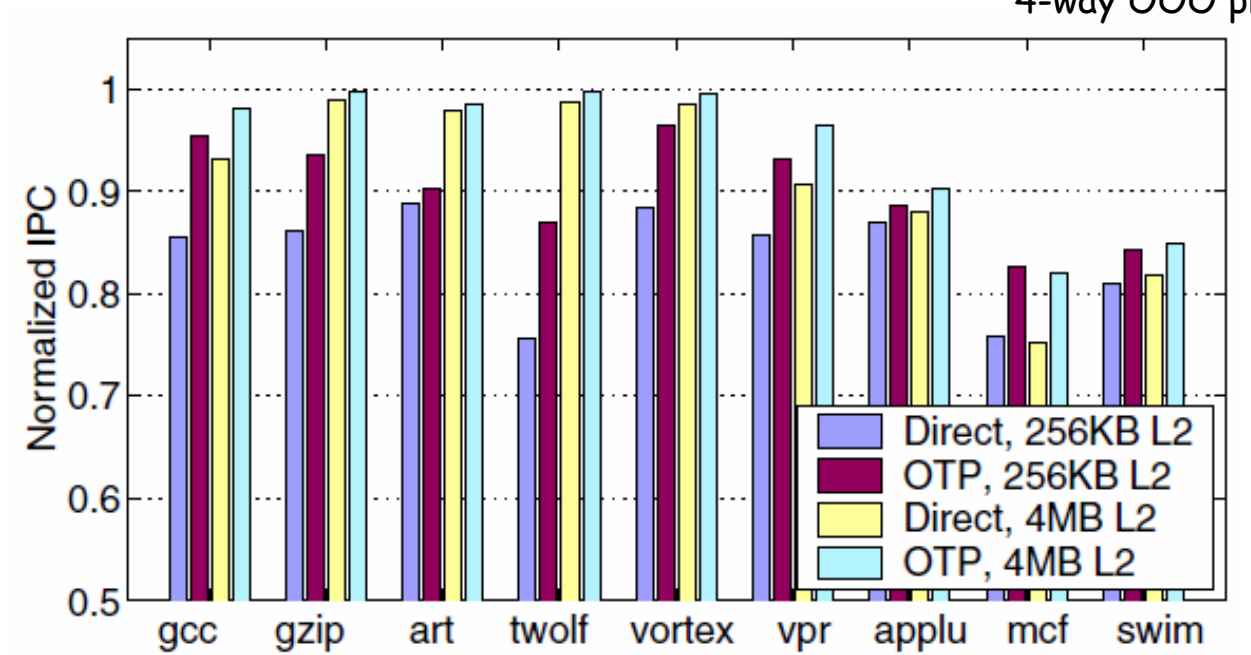


G. Edward Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas,"Efficient Memory Integrity Verification and Encryption for Secure Processors," Int. Symp. on Microarchitecture, pp.339-350, Dec. 2003.

OTPによる性能オーバヘッドの低減



4-way OOO processor



G. Edward Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas,"Efficient Memory Integrity Verification and Encryption for Secure Processors," Int. Symp. on Microarchitecture, pp.339-350, Dec. 2003.

安全性を向上する(まとめ)

- ポイント
  - 認証や暗号技術をプロセッサ・レベルに持ち込む!
  - システム階層での最後の砦!
  - 様々なオーバヘッド削減技術

- 今後の課題と展望
  - 終わり無き戦い(色々な攻撃があるため)
    - サイドチャネル攻撃など(ISCA'07:キャッシュ攻撃)
  - システム上位階層と協調(すべきか?)
  - 消費電力/エネルギー・オーバヘッド

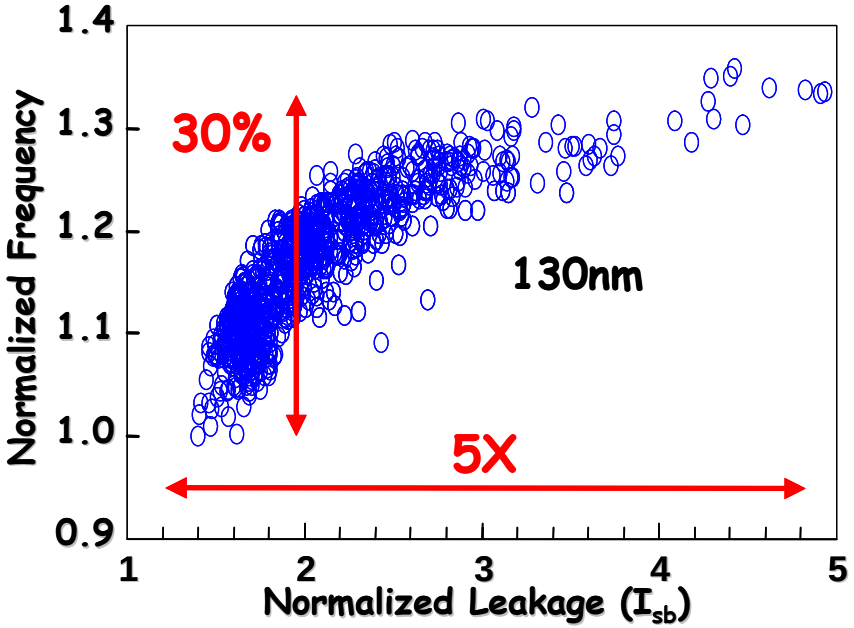
チュートリアル内容

- 信頼性(耐故障性)を向上する！
  - ソフトエラー対策
  - タイミングエラー対策
- 安全性を向上する！
  - 不正プログラムの実行防止
  - メモリデータ改ざんの防止
  - 情報漏えいの防止
- プロセスばらつきの影響を緩和する！
  - 製造後のパイプライン・チューニング
- ハードウェアバグを回避する！
  - ハードウェア・パッチング
- まとめ

プロセスばらつき

- ゲート長
- トランジスタ閾値

$$T_g \propto \frac{L_{eff} V_{dd}}{\mu (V_{dd} - V_t)^{\alpha}}$$



S. Borkar, Parameter variations and impact on circuits and microarchitecture, DAC, 2003.

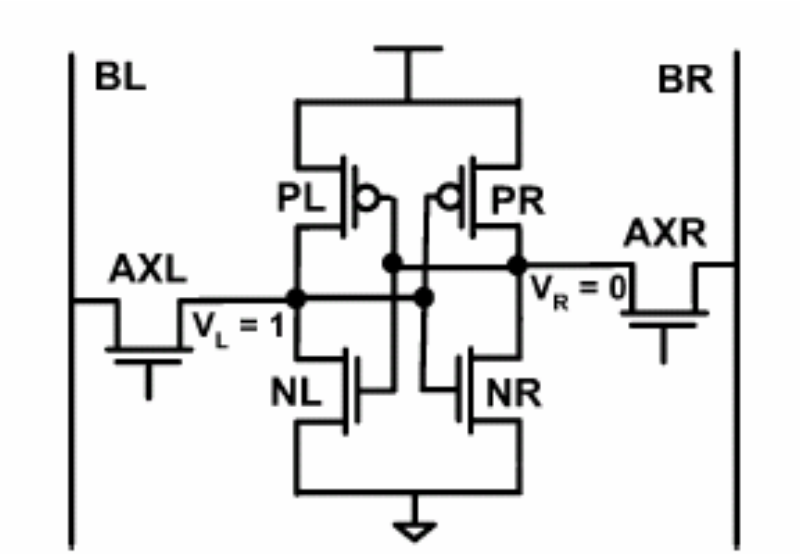
どのような影響があるのか?

- プロセスばらつきによる影響(SRAM)
  - 正しく読めない，書けない
  - 動作が遅くなる

- プロセスばらつきによる影響(Logic)
  - 設計時想定した回路遅延と製造後の回路遅延が異なる
  - つまり，クリティカル・パスが製造後でないと特定できない(想定外のパスがクリティカルになる)
  - 影響を受けやすいのは・・・
    - クリティカル・パス中の論理ゲート数が少ない
    - クリティカル・パスの数が多い

Process-Tolerant Cache Architecture

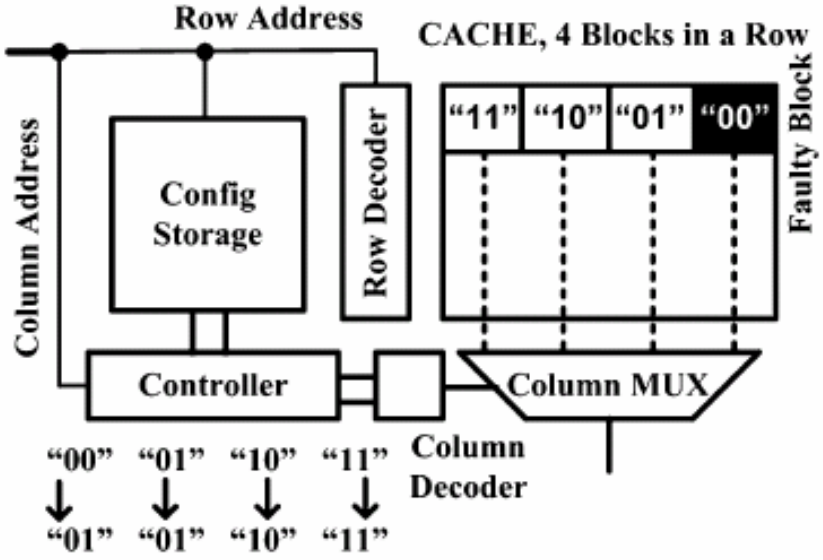
- Access Time Failure (AF)
- Read Stability Failure (RF)
- Write Stability Failure (WF)



A. Agarwal et. al., "A Process-Tolerant Cache Architecture for Improved Yield in Nanoscale Technologies," IEEE Tran. on VLSI, 2005.

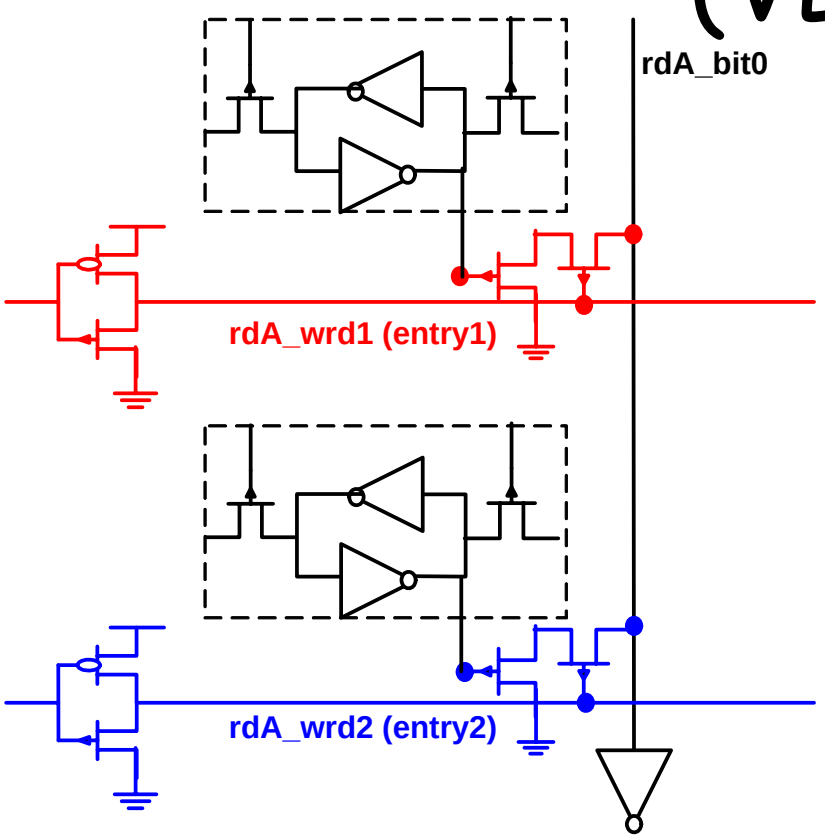
Process-Tolerant Cache Architecture

- BISTによるオンライン チェック
- Config Storageに不具合ビット箇所を記憶
- Cache Resizingにより不具合箇所の使用を回避
- キャッシュミス率は増加

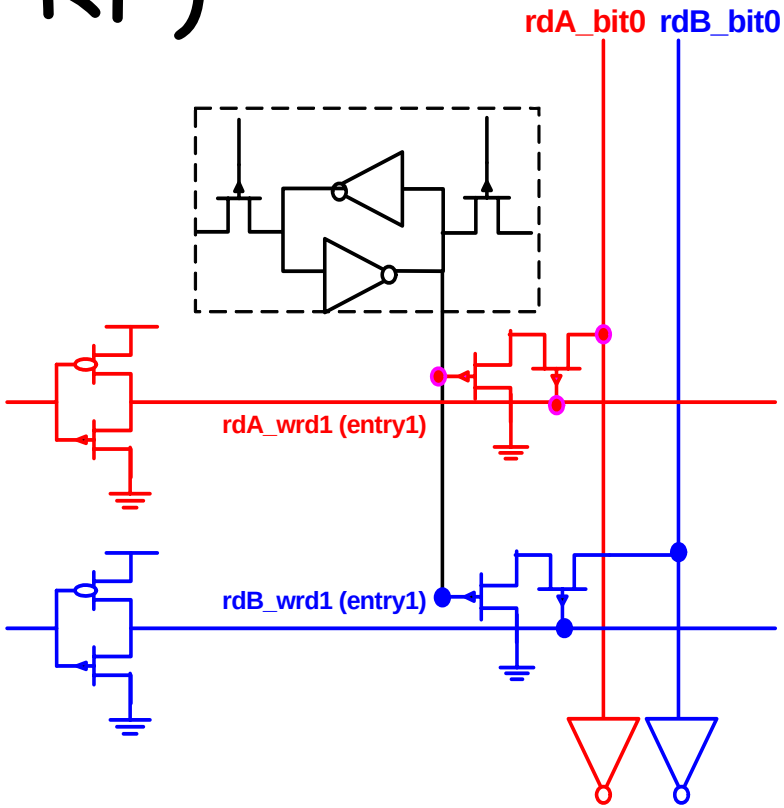


A. Agarwal et. al., "A Process-Tolerant Cache Architecture for Improved Yield in Nanoscale Technologies," IEEE Tran. on VLSI, 2005.

Variable-Latency Register File (VL-RF)



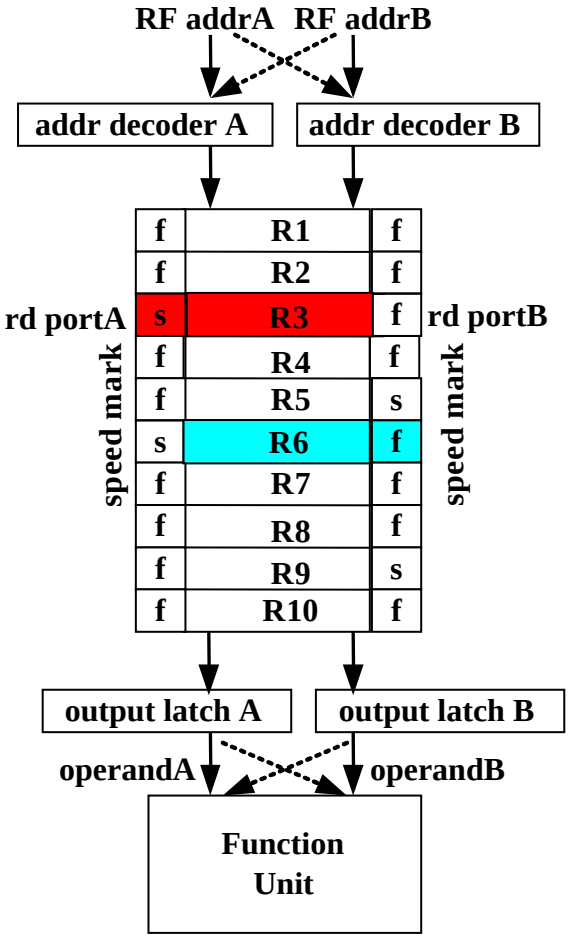
Same port reading different entries has different timing.



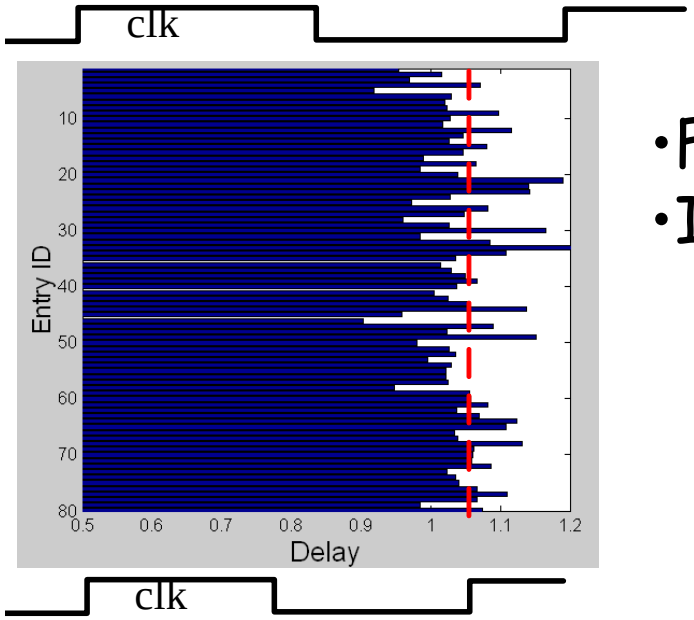
Different ports reading the same entry have different timing.

X. Liang and D. Brooks, "Mitigating the Impact of Process Variations on Processor Register Files and Execution Units," MICRO'06.

Variable-Latency Register File (VL-RF)



Original RF frequency

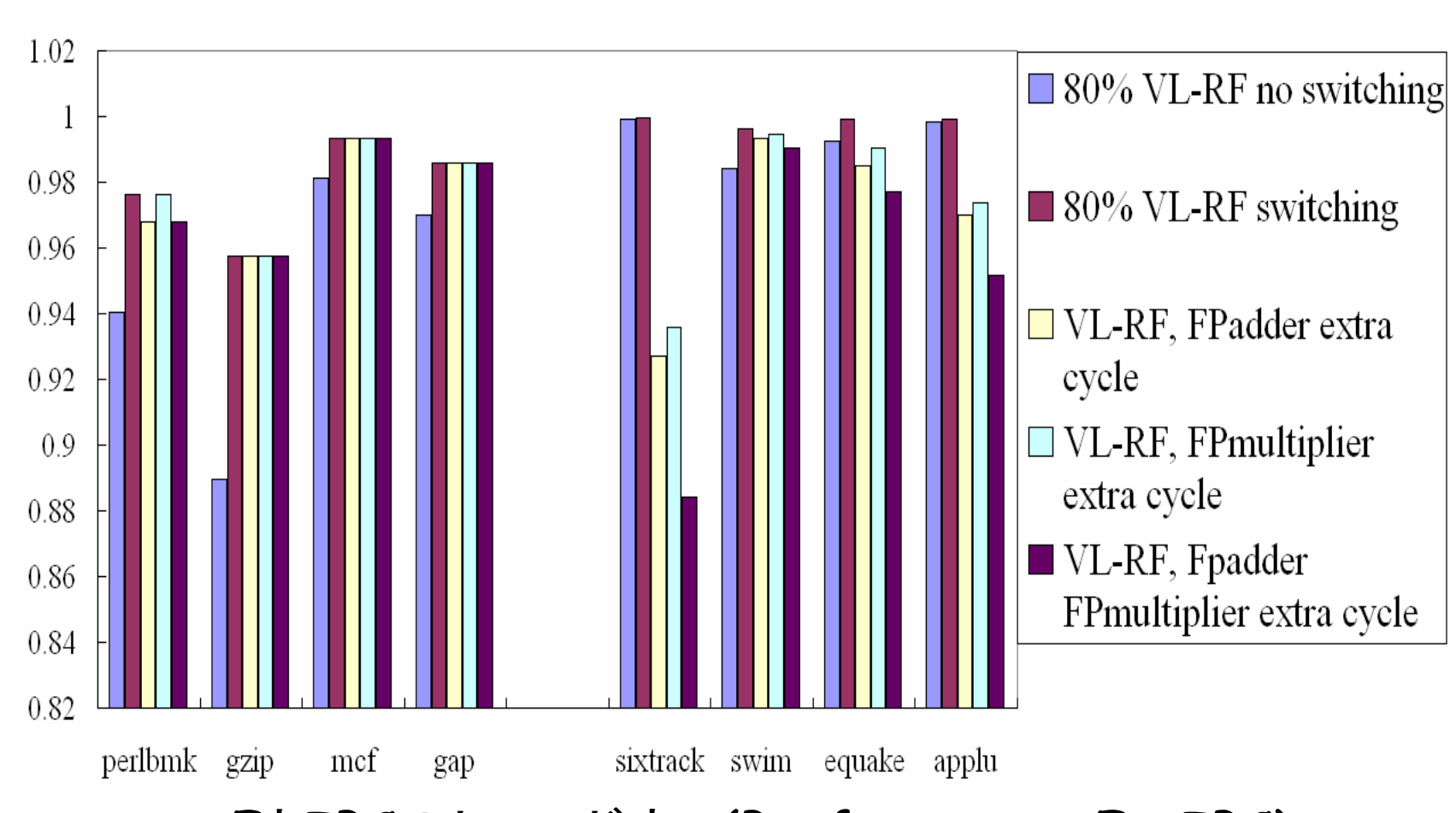


VL-RF frequency

- Fを23%改善
- IPCロスは3%
  - Port Switching
  - Port Forwarding
  - Time Borrowing
  - など

X. Liang and D. Brooks, "Mitigating the Impact of Process Variations on Processor Register Files and Execution Units," MICRO'06.

Variable-Latency Register File (VL-RF)

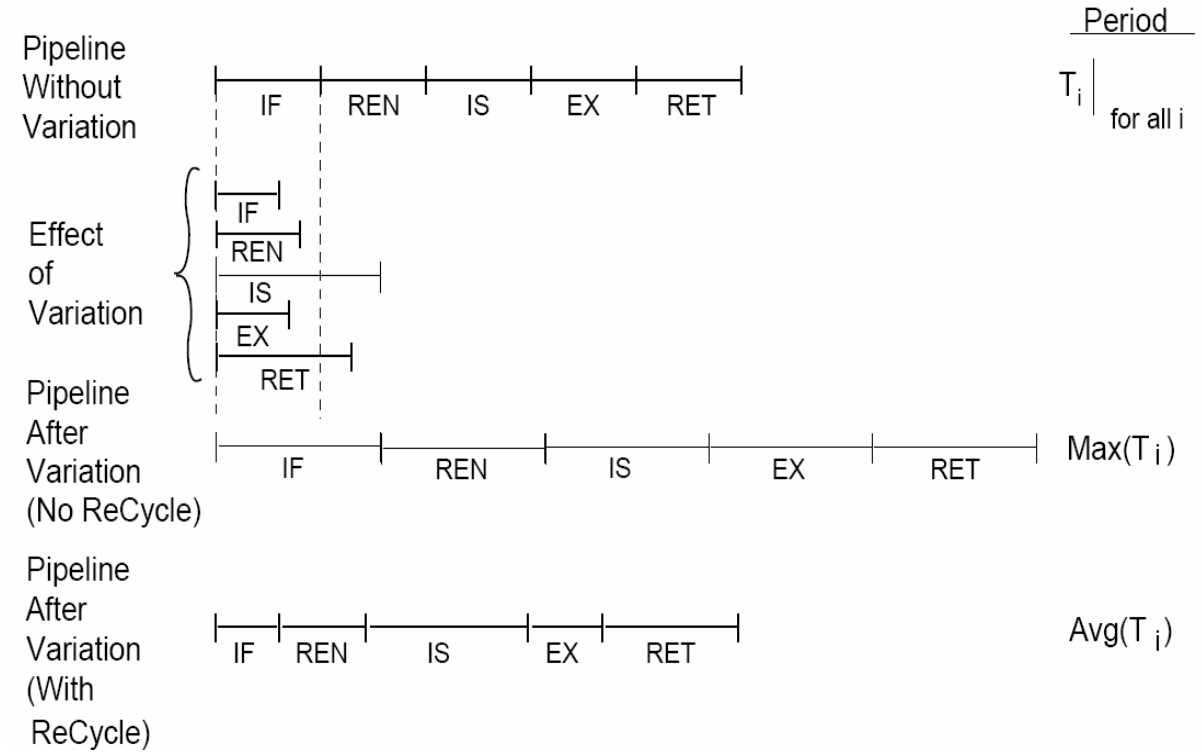


FとIPCのトレードオフ(Performance=F×IPC)

X. Liang and D. Brooks, "Mitigating the Impact of Process Variations on Processor Register Files and Execution Units," MICRO'06.

ReCycle: Pipeline Adaptation (ISCA'07)

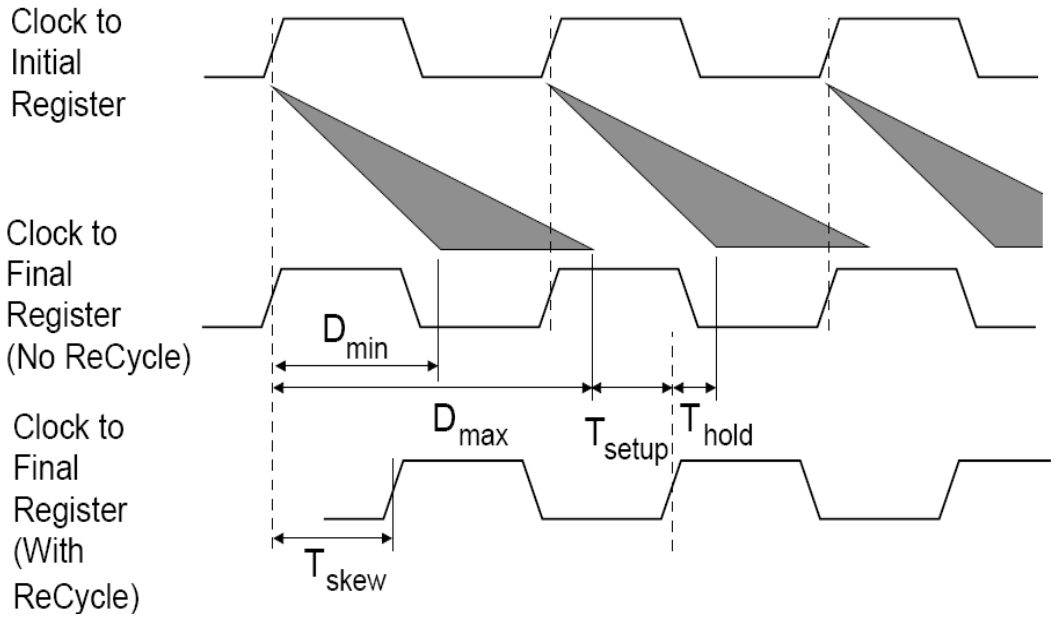
- パイプライン・レベルのCycle-Time Stealing
- ステージ間アンバランスを解消



A. Tiwari, S. R. Sarangi, and J.Torrellas, "ReCycle: Pipeline Adaptation to Tolerate Process Variation ," ISCA'07.

Cycle-Time Stealing

- チップ製造後に各ステージのレイテンシを測定
- サイクル時間を最小にするTskewを決定
  - Linear Program formulation



Setup constraint:

$$\partial_f - \partial_i + T_{cp} \geq D_{max} + T_{setu}$$

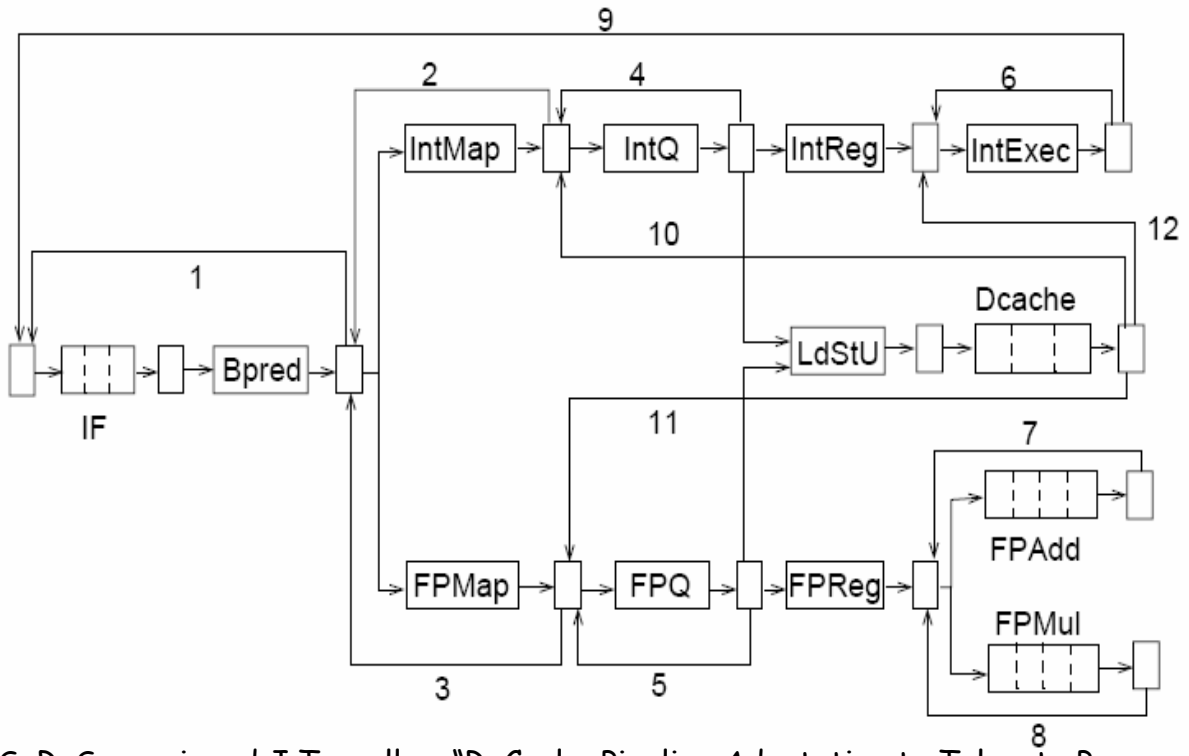
Hold constraint

$$\partial_f - \partial_i \leq D_{min} - T_{hold}$$

A. Tiwari, S. R. Sarangi, and J.Torrellas, "ReCycle: Pipeline Adaptation to Tolerate Process Variation ," ISCA'07.

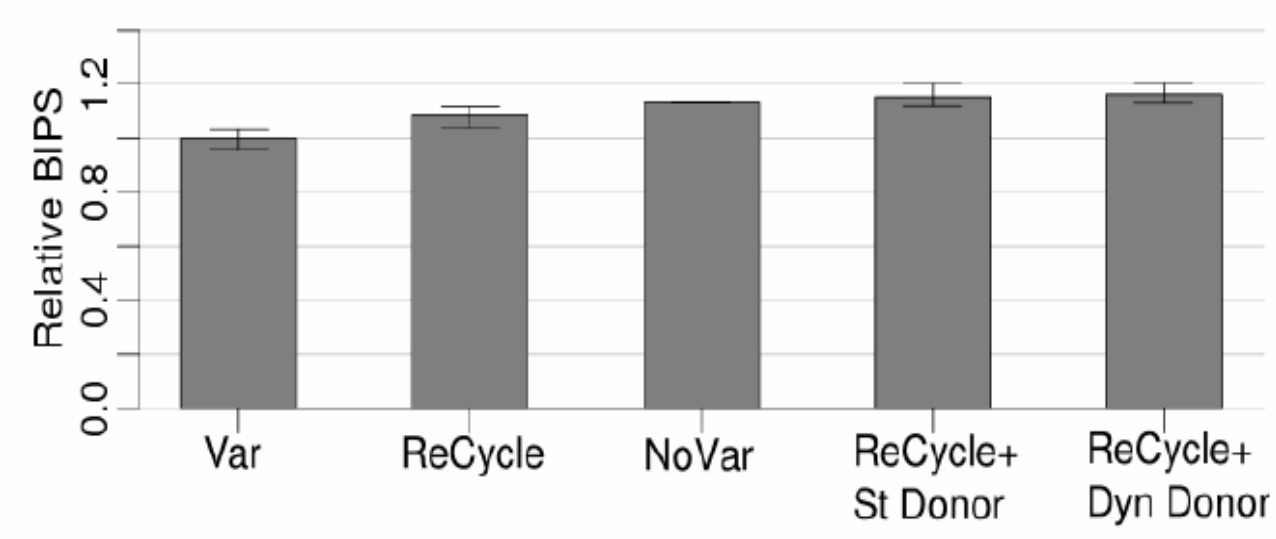
Donor Stages

- Pipeline Loopを定義
- Critical LoopにDonor Stage (空ステージ)を挿入→積極的なCycle-Time Stealing



A. Tiwari, S. R. Sarangi, and J.Torrellas, "ReCycle: Pipeline Adaptation to Tolerate Process Variation ," ISCA'07.

Performance



- Perfomance of ReCycle is 9% higher than Var
- ReCycle does not catch up with NoVar
- Performance of ReCycle+StDonor is higher than NoVar

FとIPCのトレードオフ(Performance=F×IPC)

A. Tiwari, S. R. Sarangi, and J.Torrellas, "ReCycle: Pipeline Adaptation to Tolerate Process Variation ," ISCA'07.

プロセスばらつきの影響を緩和する（まとめ）

- ポイント
  - チップ製造後の調整機能
  - FとIPCのトレードオフ
- 今後の課題と展望
  - そもそも，ばらつきの影響を受けないアーキテクチャは?（SWoPP'06Sato）
  - ベンダーからの情報提供/評価環境の構築
  - ばらつきを考慮した低消費電力/エネルギー化技術

チュートリアル内容

- 信頼性(耐故障性)を向上する！
  - ソフトエラー対策
  - タイミングエラー対策
- 安全性を向上する！
  - 不正プログラムの実行防止
  - メモリデータ改ざんの防止
  - 情報漏えいの防止
- プロセスばらつきの影響を緩和する！
  - 製造後のパイプライン・チューニング
- ハードウェアバグを回避する！
  - ハードウェア・パッチング
- まとめ

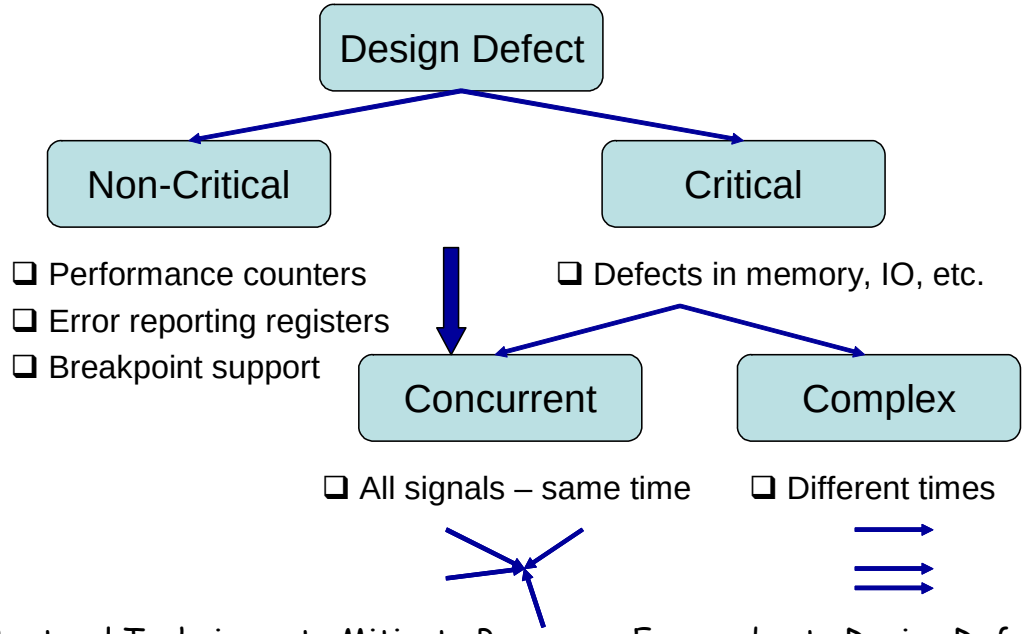
商用プロセッサにおけるバグ

1994 Pentium defect costs Intel $475 million

1999 Defect leads to stoppage in shipping Pentium III servers

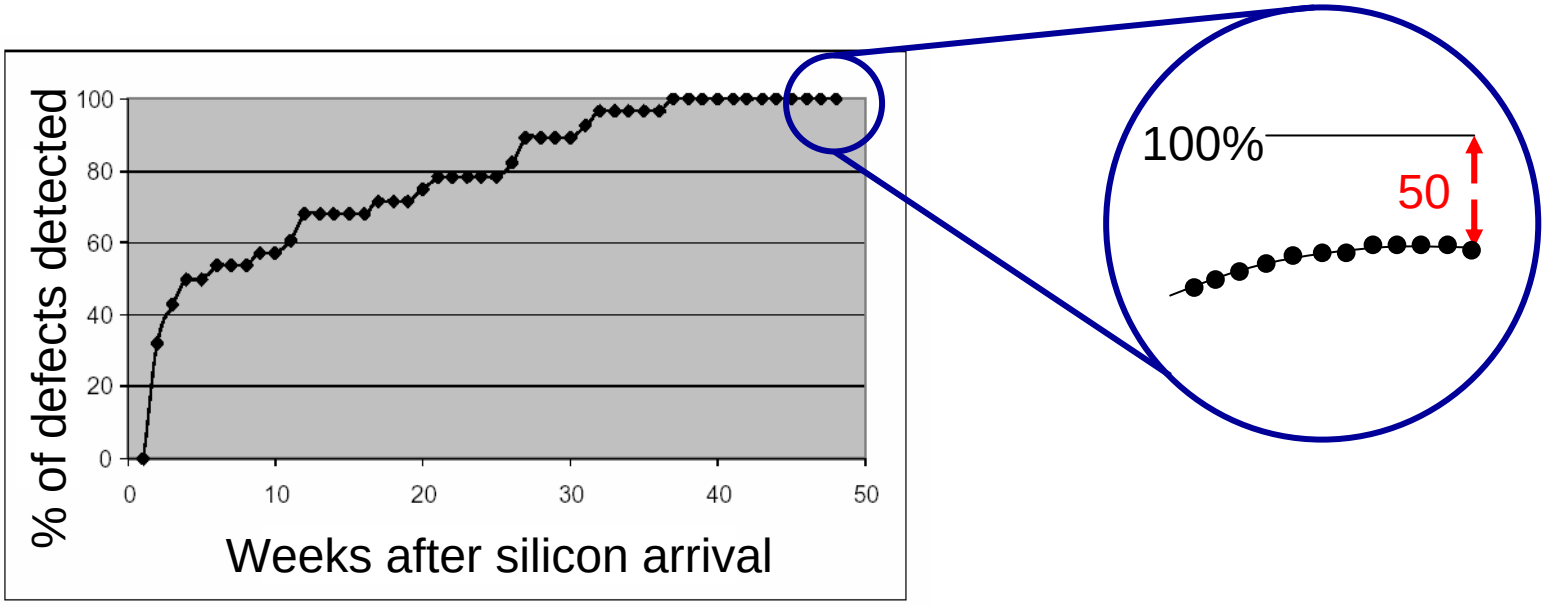
2004 AMD Opteron defect leads to data loss

2005 A version of Itanium 2 recalled



J. Torrellas, "Novel Architectural Techniques to Mitigate Processor Errors due to Design Defects and Parameter Variation," CoolChips'07.

Defects in Deployed Systems



- We studied public domain errata documents for 10 current processors

  Intel Pentium III, IV, M, and Itanium I and II

  AMD K6, Athlon, Athlon 64

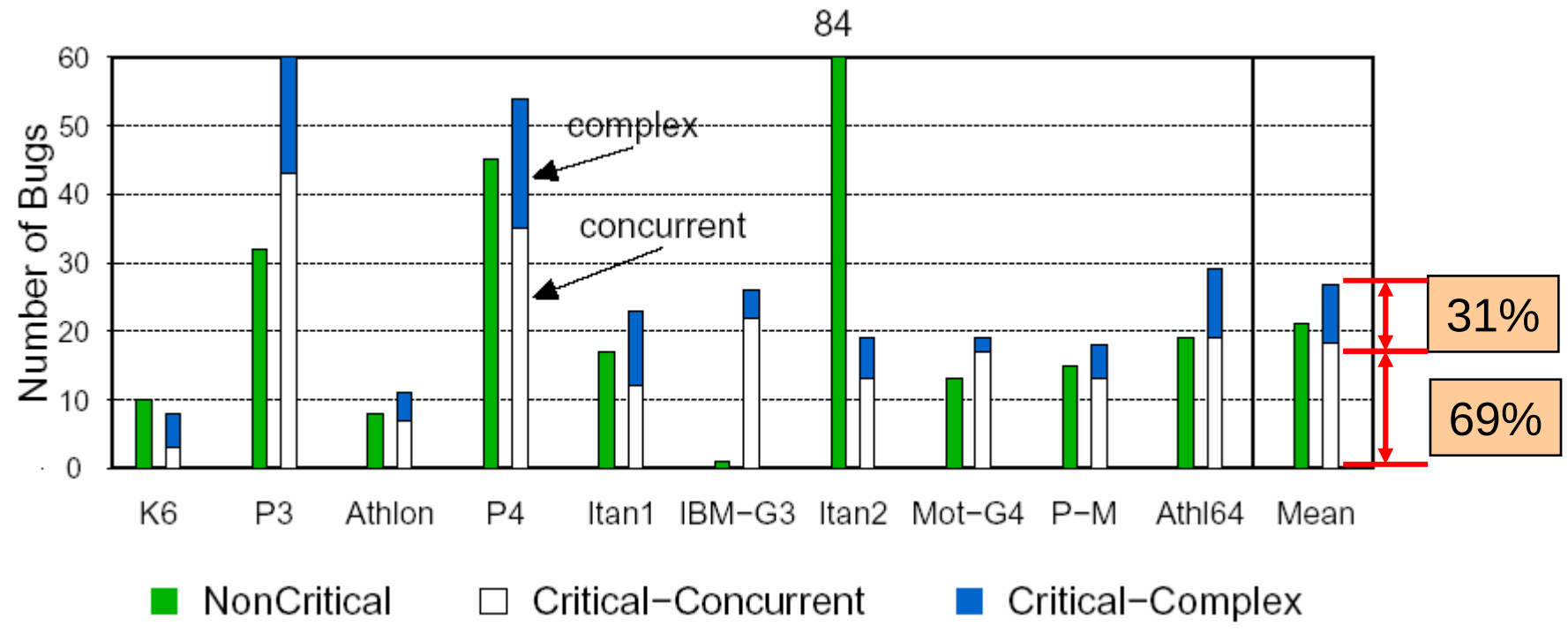
  IBM G3 (PPC 750 FX), MOT G4 (MPC 7457)

どのようなバグが，どの程度あるのか?

商用プロセッサのエラーレポートを調査

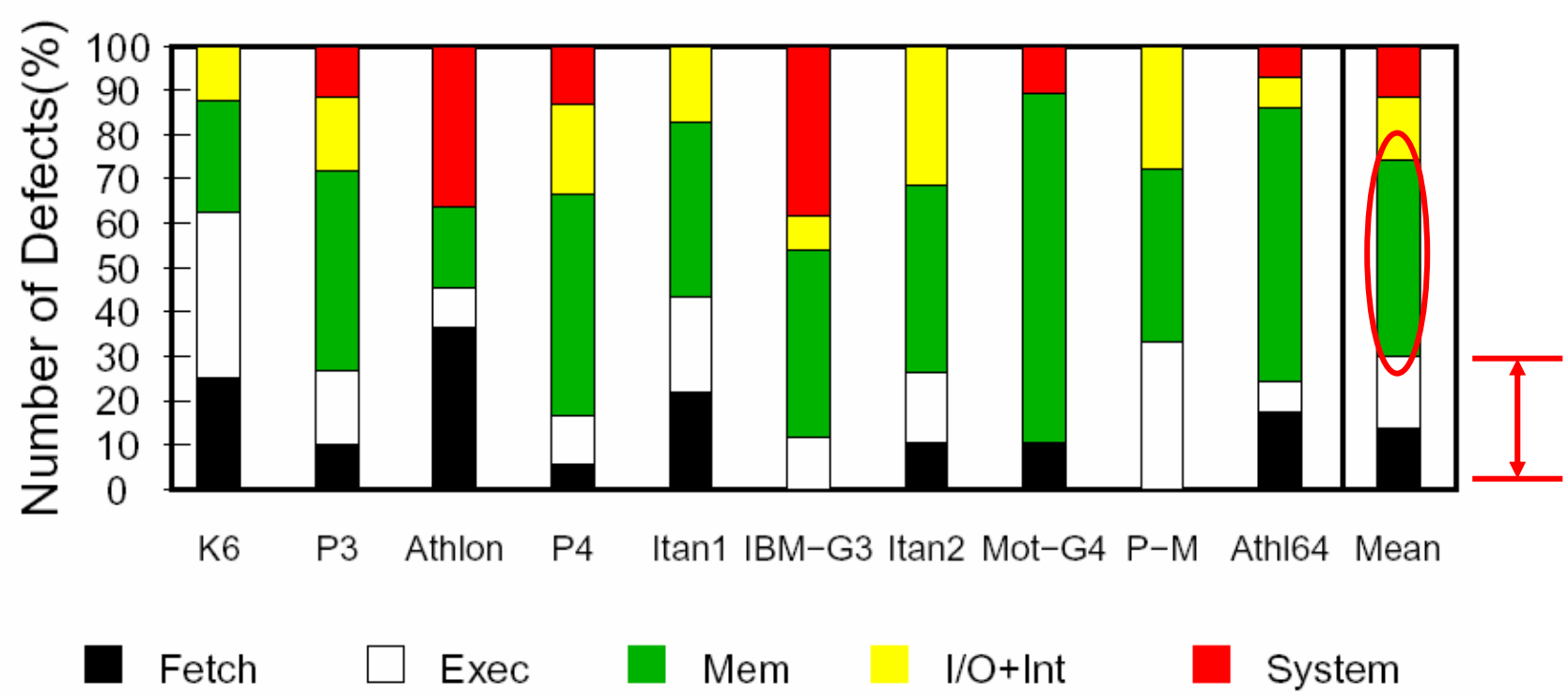
Intel Pentium III, IV, M, and Itanium I and II, AMD K6, Athlon, Athlon 64

IBM G3 (PPC 750 FX), MOT G4 (MPC 7457)



J. Torrellas, "Novel Architectural Techniques to Mitigate Processor Errors due to Design Defects and Parameter Variation," CoolChips'07.

どこにバグが潜んでいるのか?

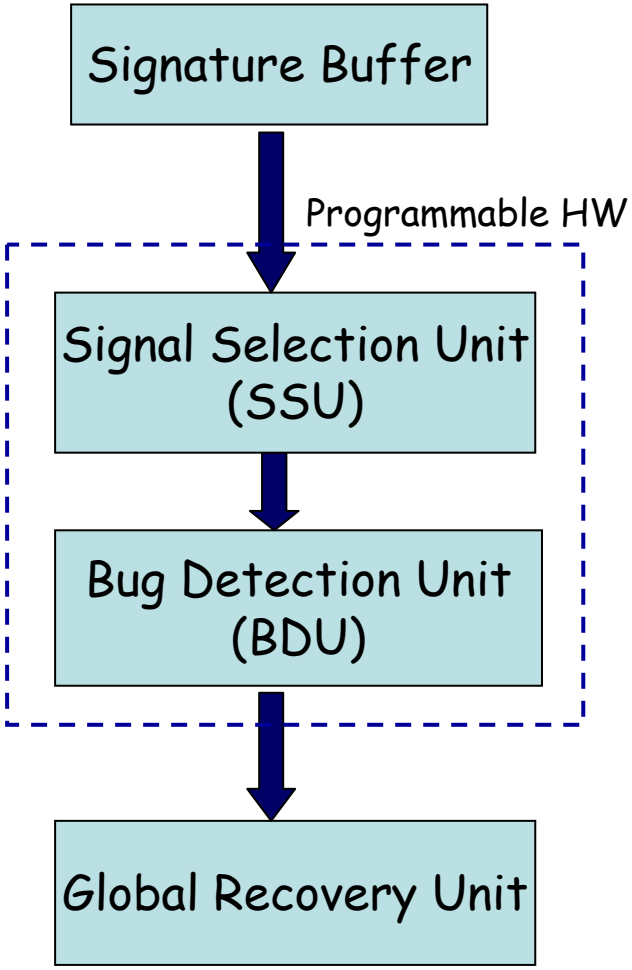


- プロセッサコアのバグは比較的少ない
- メモリ周りのバグが多い

J. Torrellas, "Novel Architectural Techniques to Mitigate Processor Errors due to Design Defects and Parameter Variation," CoolChips'07.

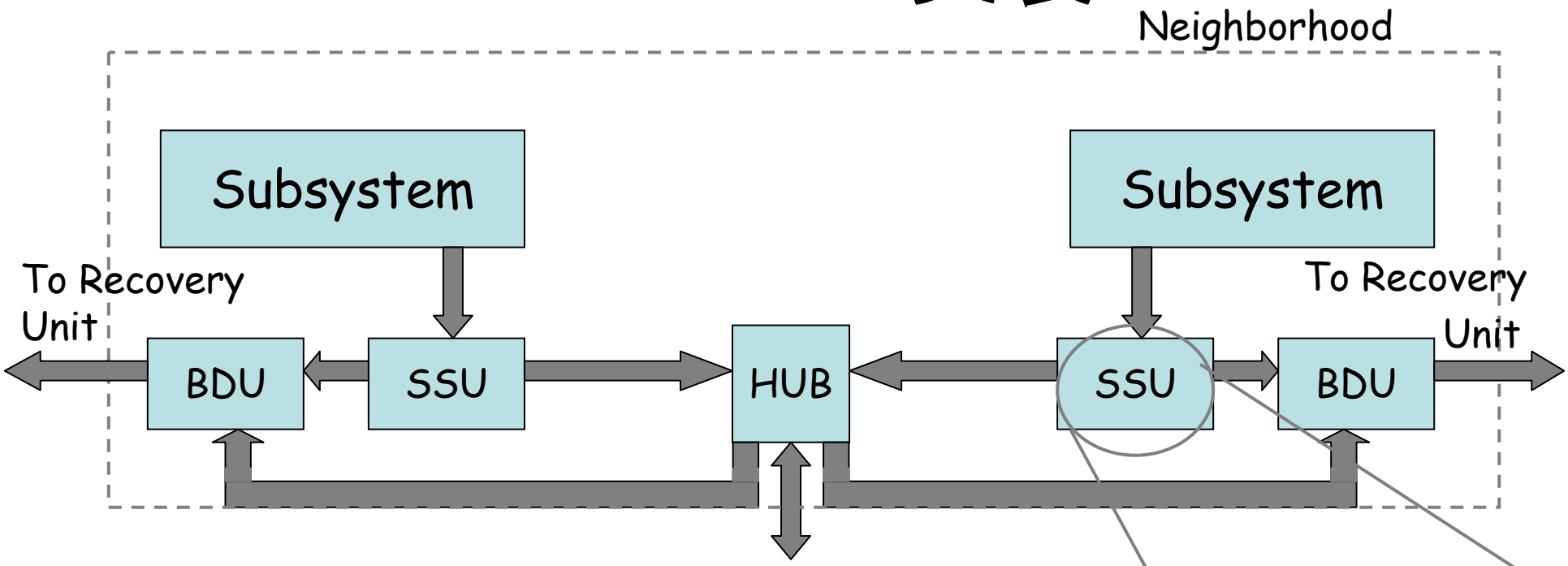
Phoenix: Hardware Patching

- Concurrent Defect（CD）に着目
- イベント信号の状態を監視すれば検出可能
  - e.g. L1 miss, L2 flush, and Power Management ON
- バグが見つかったら・・・
  - ベンダーからCD発生条件に関する情報入手(Defect Signature)
  - SSUとBDRを再構成
  - CE発生条件が成立したらリカバリ
    - pipeline flush, recovery handler



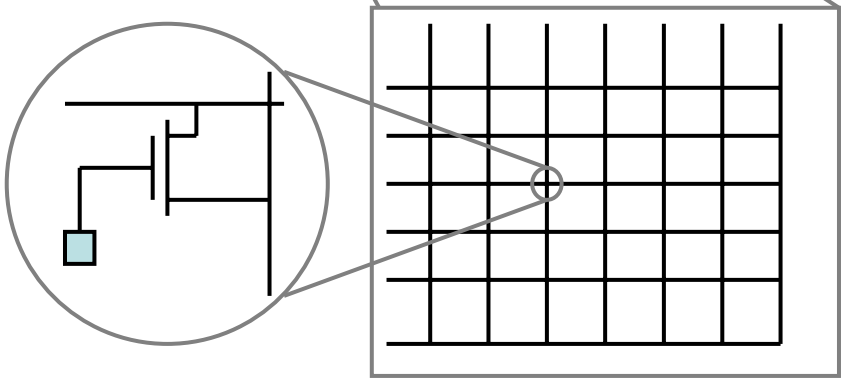
S. R. Sarangi, A. Tiwari, and J. Torrellas, "Phoenix: Detecting and Recovering from Permanent Processor Design Bugs with Programmable Hardware," MICRO'06.

Phoenixの実装



Examples of Subsystems

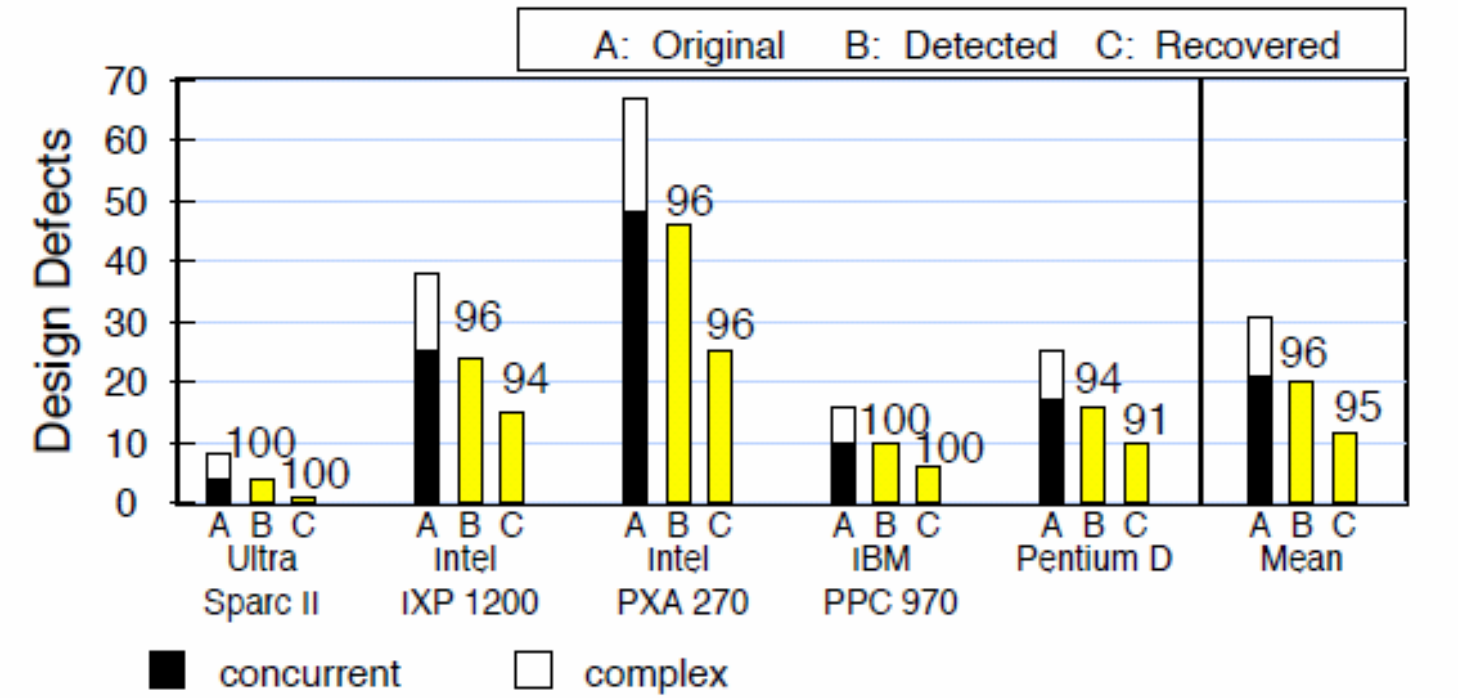
Inst. Cache	FP ALU	Virtual Mem.
Fetch Unit	L1 Cache	IO Cntrl.



S. R. Sarangi, A. Tiwari, and J. Torrellas, "Phoenix: Detecting and Recovering from Permanent Processor Design Bugs with Programmable Hardware," MICRO'06.

Phoenixのバグ検出/回復能力

- 10種類のプロセッサをベースにPhonixを設計
- 他5種類のプロセッサでバグ検出/回復能力を評価



S. R. Sarangi, A. Tiwari, and J. Torrellas, "Phoenix: Detecting and Recovering from Permanent Processor Design Bugs with Programmable Hardware," MICRO'06.

ハードウェアバグを回避する(まとめ)

- ポイント
  - ハードウェア・バグの分類
    - ここでは，Concurrent Defectに着目
  - 柔軟性の維持

- 今後の課題と展望
  - CD以外のバグへの対応
  - ソフトウェア・バグ対策への展開
    - ソフトウェア生産性を考慮したアーキテクチャサポート
    - 特にCMPでは重要に!?

チュートリアル内容

- 信頼性（耐故障性）を向上する！
  - ソフトエラー対策
  - タイミングエラー対策
- 安全性を向上する！
  - 不正プログラムの実行防止
  - メモリデータ改ざんの防止
  - 情報漏えいの防止
- プロセスばらつきの影響を緩和する！
  - 製造後のパイプライン・チューニング
- ハードウェアバグを回避する！
  - ハードウェア・パッチング
- まとめ

真の「ディペンダブル」を目指して

- 本チュートリアルでは・・・
  - 「ディペンダブル」プロセッサの研究動向
    - ソフトエラー対策
    - セキュリティ対策
    - プロセスばらつき対策
    - ハードウェア・バグ対策
- アーキテクチャ屋は大変・・・
  - 下位階層（回路/デバイス）の揺らぎ
  - 上位階層（ソフトウェア）の揺らぎ
  - 是非，アーキテクチャの世界へ!