自己防衛能力を持つプロセッサ・システムの実現に向けて

井上, 弘士
九州大学大学院システム情報科学研究院

出版情報：SLRC プレゼンテーション, 2006-07-19. 九州大学システムLSI研究センター
バージョン：
権利関係：

自己防衛能力を持つ プロセッサ・システムの実現に向けて

九州大学大学院システム情報科学研究院

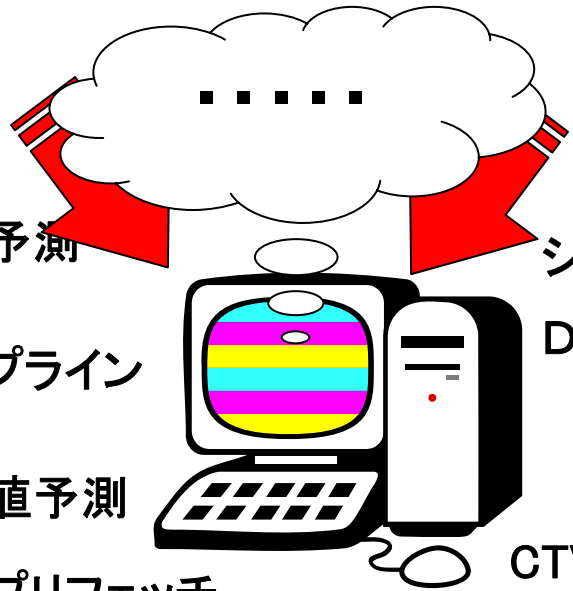
井上弘士（いのうえこうじ）

inoue@i.kyushu-u.ac.jp

プロセッサ・アーキテクチャ開発の現状

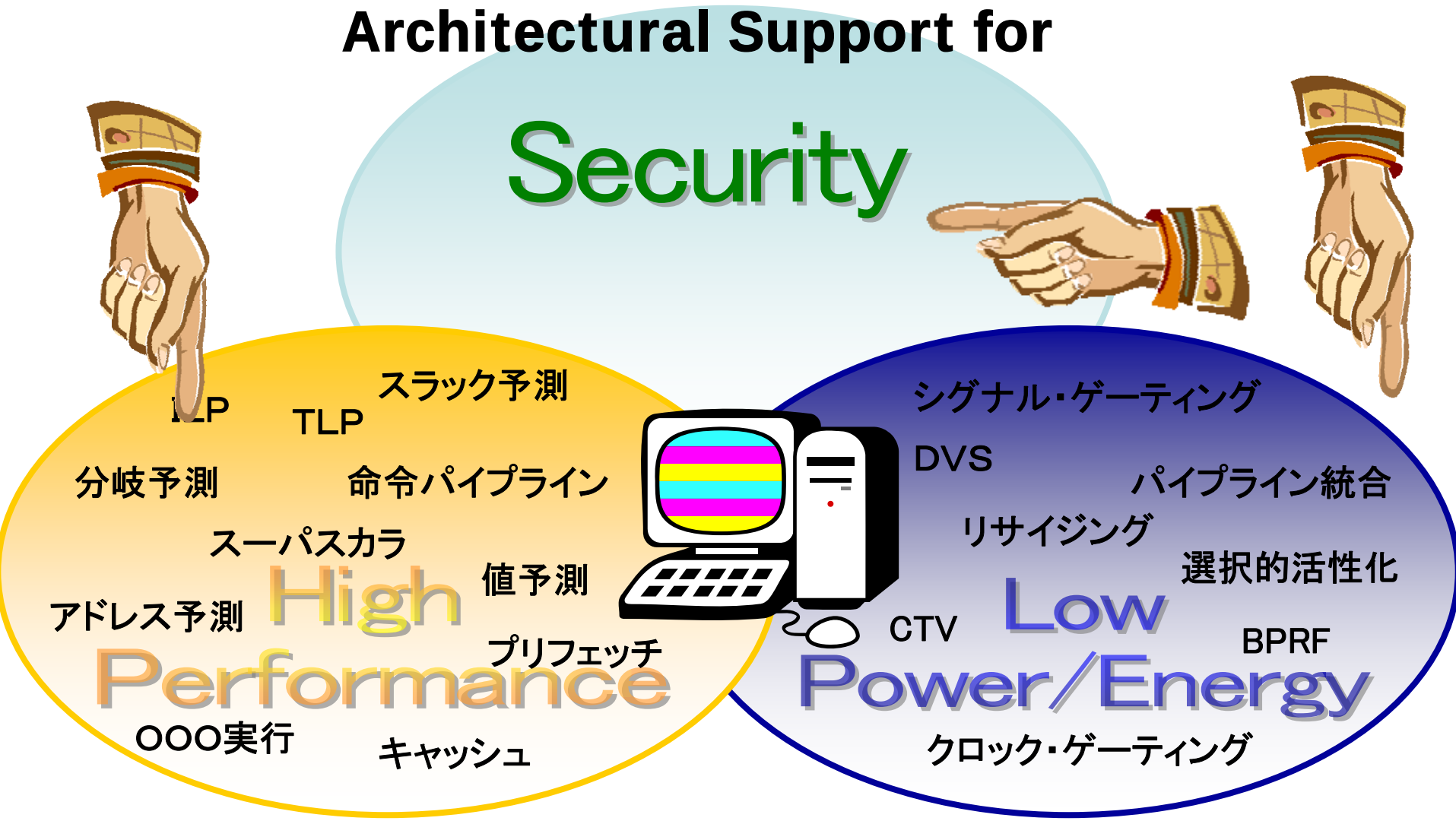
正規
プログラム

アタック
コード



ILP
TLP
スラック予測
シグナル・ゲーティング
分岐予測
命令パイプライン
DVS
パイプライン統合
スーパスカラ
リサイジング
選択的活性化
アドレス予測
High
値予測
Low
CTV
BPRF
プリフェッチ
Performance
Power/Energy
OOO実行
キャッシュ
クロック・ゲーティング

本研究で目指すところ・・・

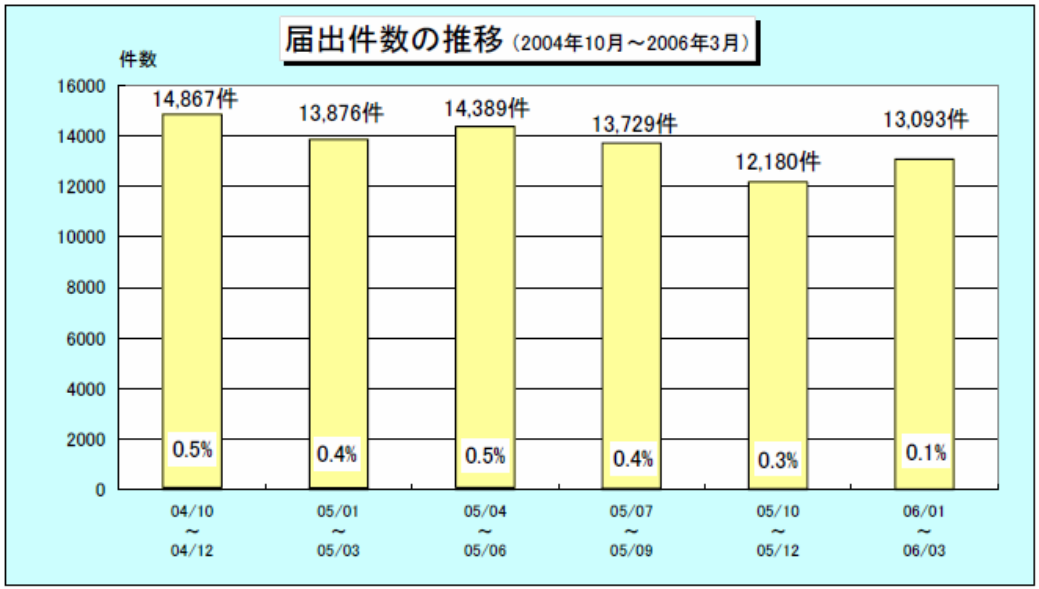

Architectural Support for
Security
スラック予測
TLP
TLP
分岐予測
命令パイプライン
スーパスカラ
High
Performance
値予測
アドレス予測
プリフェッチ
OOO実行
キャッシュ
シグナル・ゲーティング
DVS
パイプライン統合
リサイジング
選択的活性化
CTV
Low
Power/Energy
BPRF
クロック・ゲーティング

研究内容紹介

- 不正プログラムの実行を防止する！
  - バッファ・オーバフロー攻撃の検出
  - 動的なプログラム認証
- メモリデータの改竄を検出する！
  - メモリ整合性検証を前提とした高性能化

コンピュータ・ウィルス問題（どの程度，ウィルス被害があるのか？）

- 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

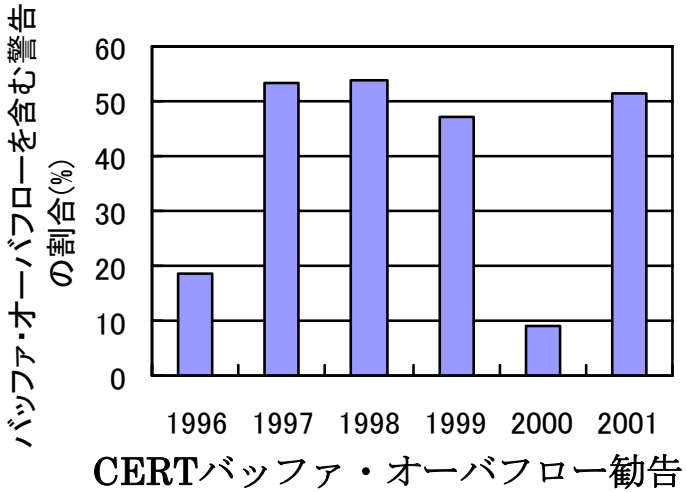


IPA2006 年第1 四半期[1 月～3 月]コンピュータウイルス届出状況より
http://www.ipa.go.jp/security/txt/2006/documents/2006q1-v.pdf

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

バッファ・オーバフロー攻撃

- バッファ・オーバフロー
  - 最も多く活用される脆弱性の1つ
  - Blaster@2003, CodeRed@2001



CERTバッファ・オーバフロー勧告

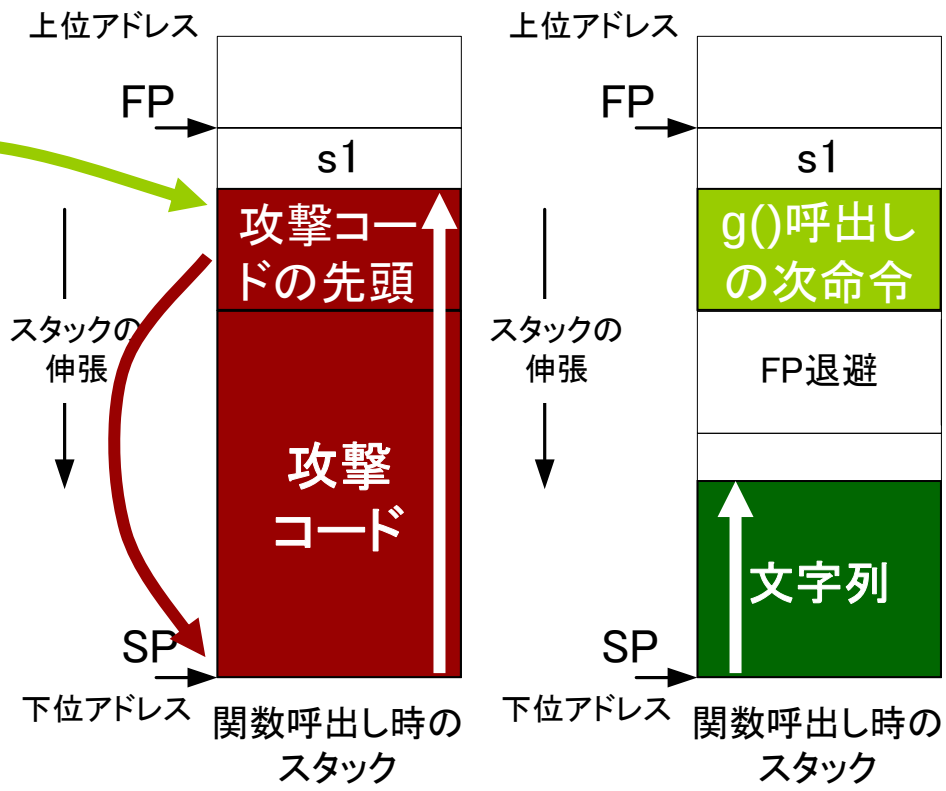
R.B.Lee, D.K.Karig, J.P.McGregor, and Z.Shi, “Enlisting Hardware Architecture to Thwart Malicious Code Injection,” Proc. of the Int. Conf. on Security in Pervasive Computing, Mar. 2003.

- 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存在（strcpyなど）
  - スタックの破壊（スタック・スマッシング）
    - 悪質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
    - 改ざん後の戻りアドレスをPCへ設定

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

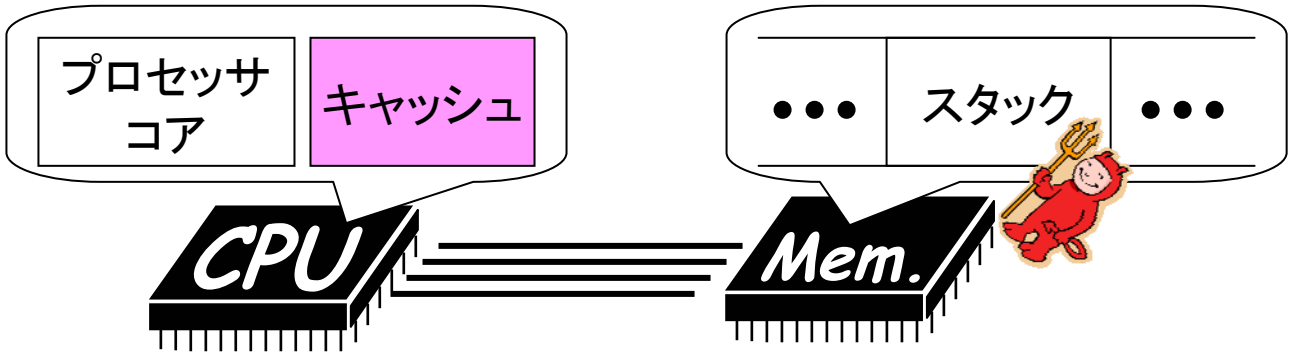
```
int f ( ) {
  …
  g (s1);
  …
}

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

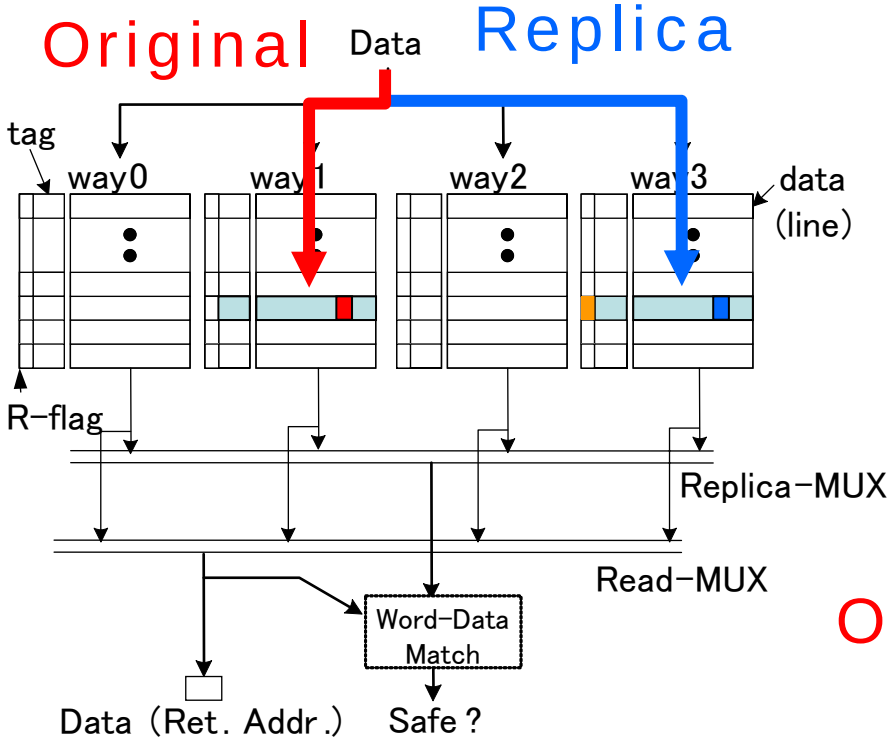
動的な戻りアドレス改ざん検出 ～Secure Cache: SCache～



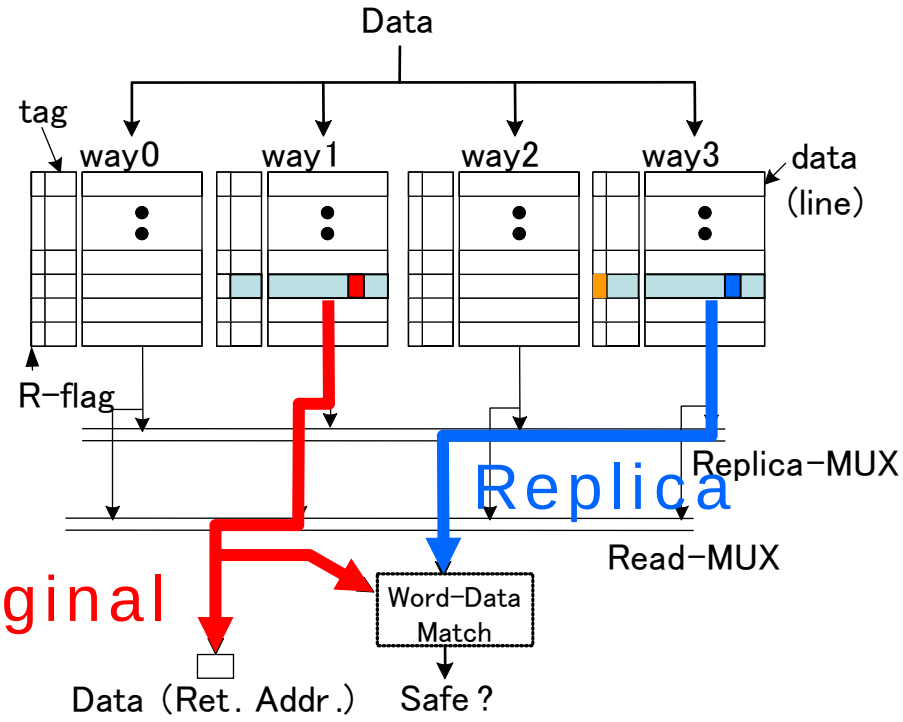
- 問題点:
  - スタックに書込んだ戻りアドレスが改ざんされる
- 解決策:
  - キャッシュで戻りアドレスを保護しよう！
- 手段:
  - 戻りアドレス書込み時に複製(レプリカ・ライン)を作成
  - 戻りアドレス読出し時に複製と比較
  - 不一致であれば戻りアドレス改ざん発生と判定

SCacheの動作

戻りアドレス書込み時

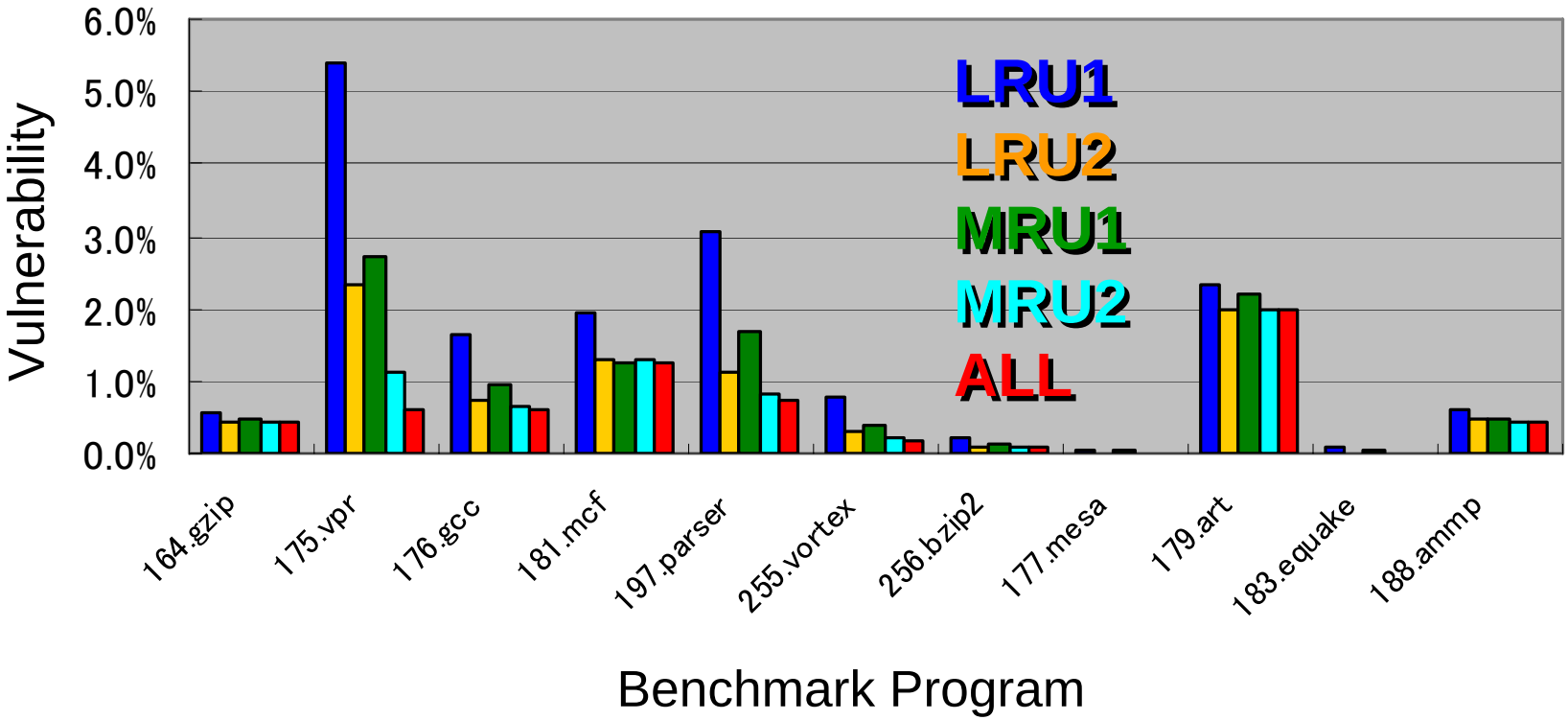


戻りアドレス読出し時

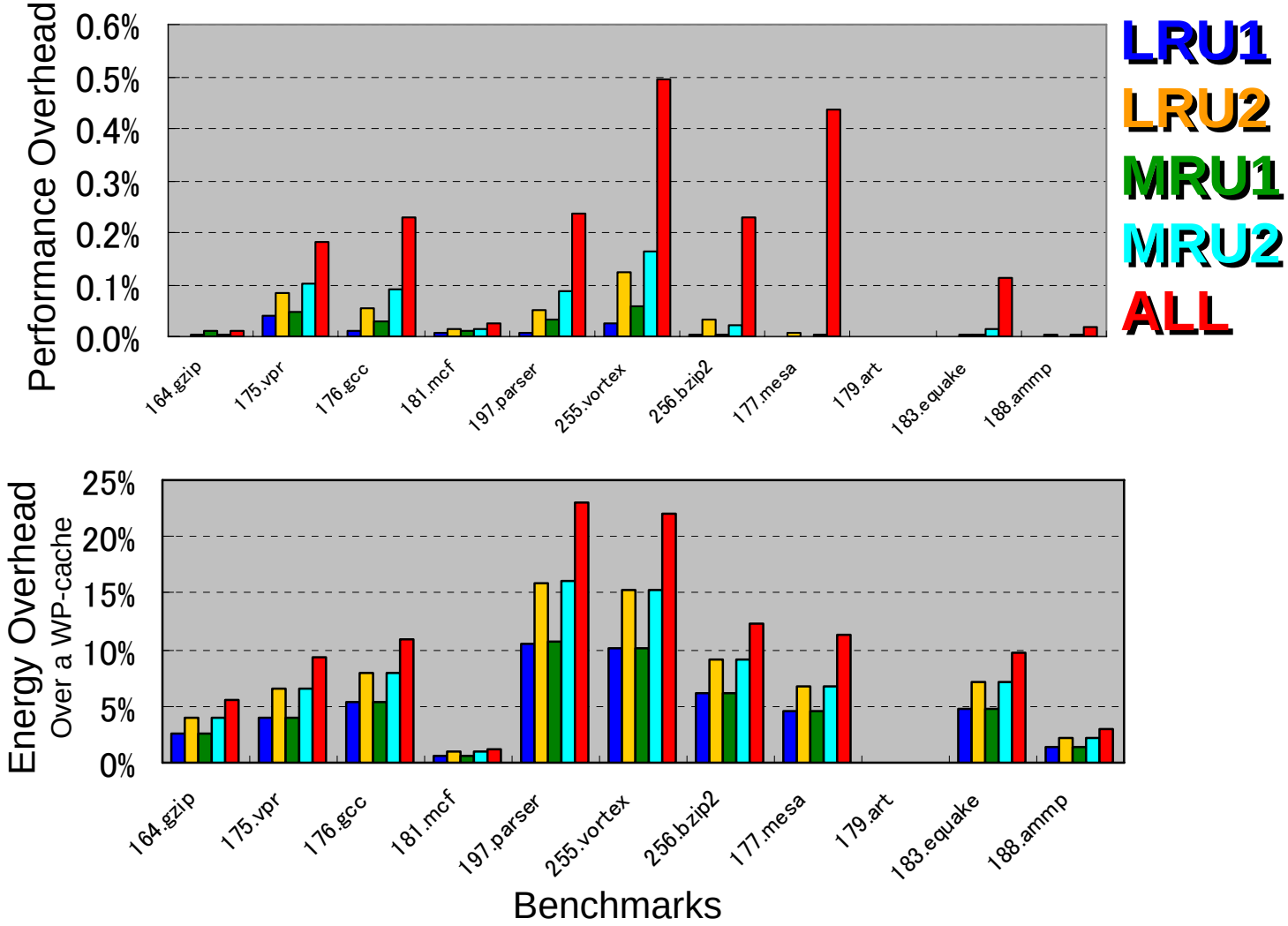


キャッシュ・ヒット時を想定

安全性に関する評価


Vulnerability
6.0%
5.0%
4.0%
3.0%
2.0%
1.0%
0.0%
LRU1
LRU2
MRU1
MRU2
ALL
164.gzip
175.vpr
176.gcc
181.mcf
197.parser
255.vortex
256.bzip2
177.mesa
179.art
183.equake
188.ammp
Benchmark Program

性能/消費エネルギー・オーバヘッド



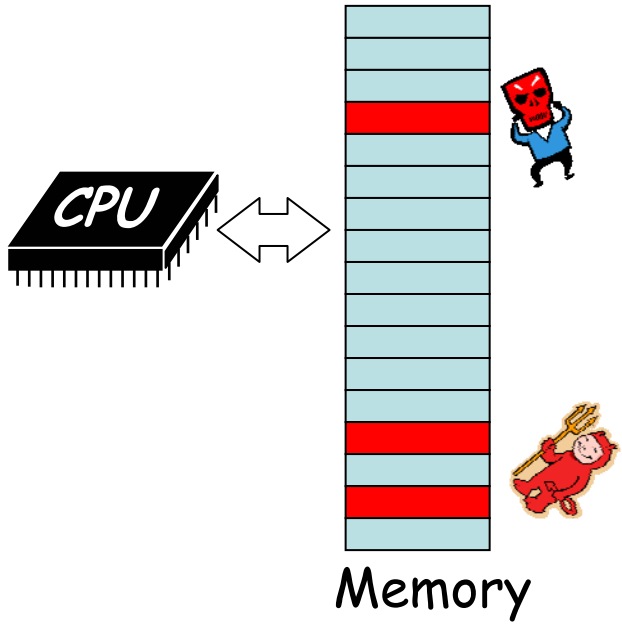
研究内容紹介

- 不正プログラムの実行を防止する！
  - –バッファ・オーバフロー攻撃の検出
  - –動的なプログラム認証

- メモリデータの改竄を検出する！
  - –メモリ整合性検証を前提とした高性能化

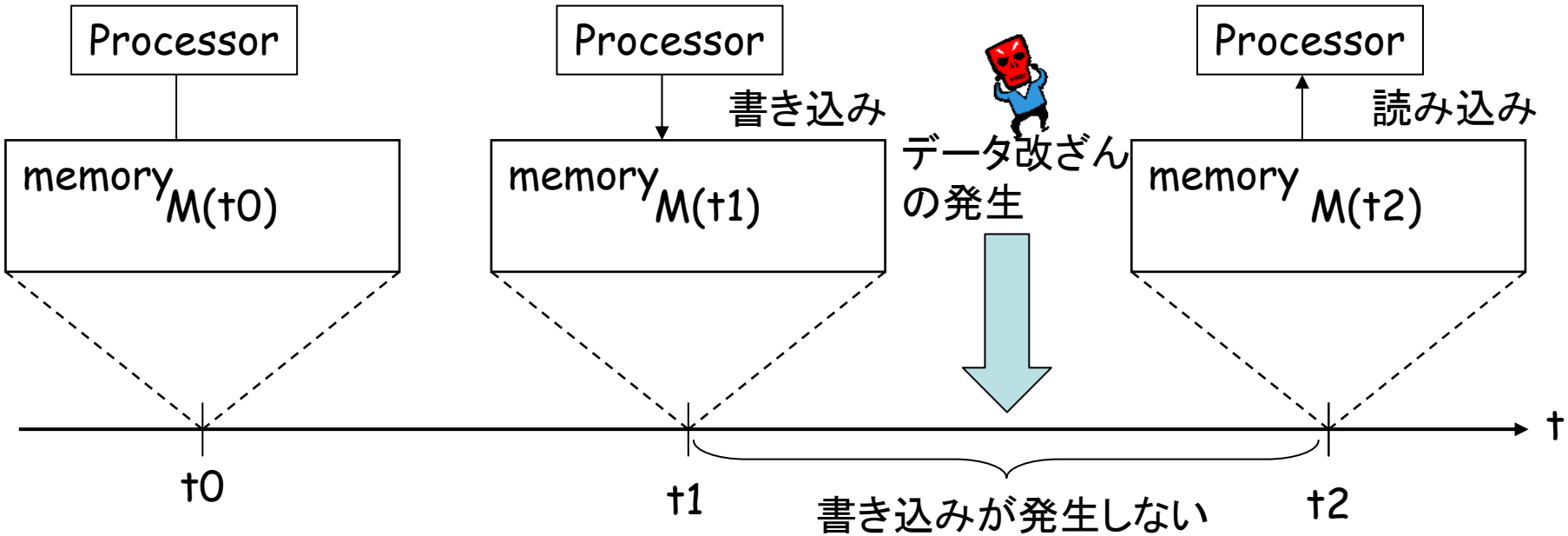
メモリデータの改竄

- Spoofing Attack
  - ある偽のデータを生成し，それが本物であるかのように見せかける
- Splicing Attack
  - データ・フラグメントのコピーや並び替えを行い他メモリ領域に格納する
- Replay Attack
  - 過去に生成された有効なデータを再度入力させる



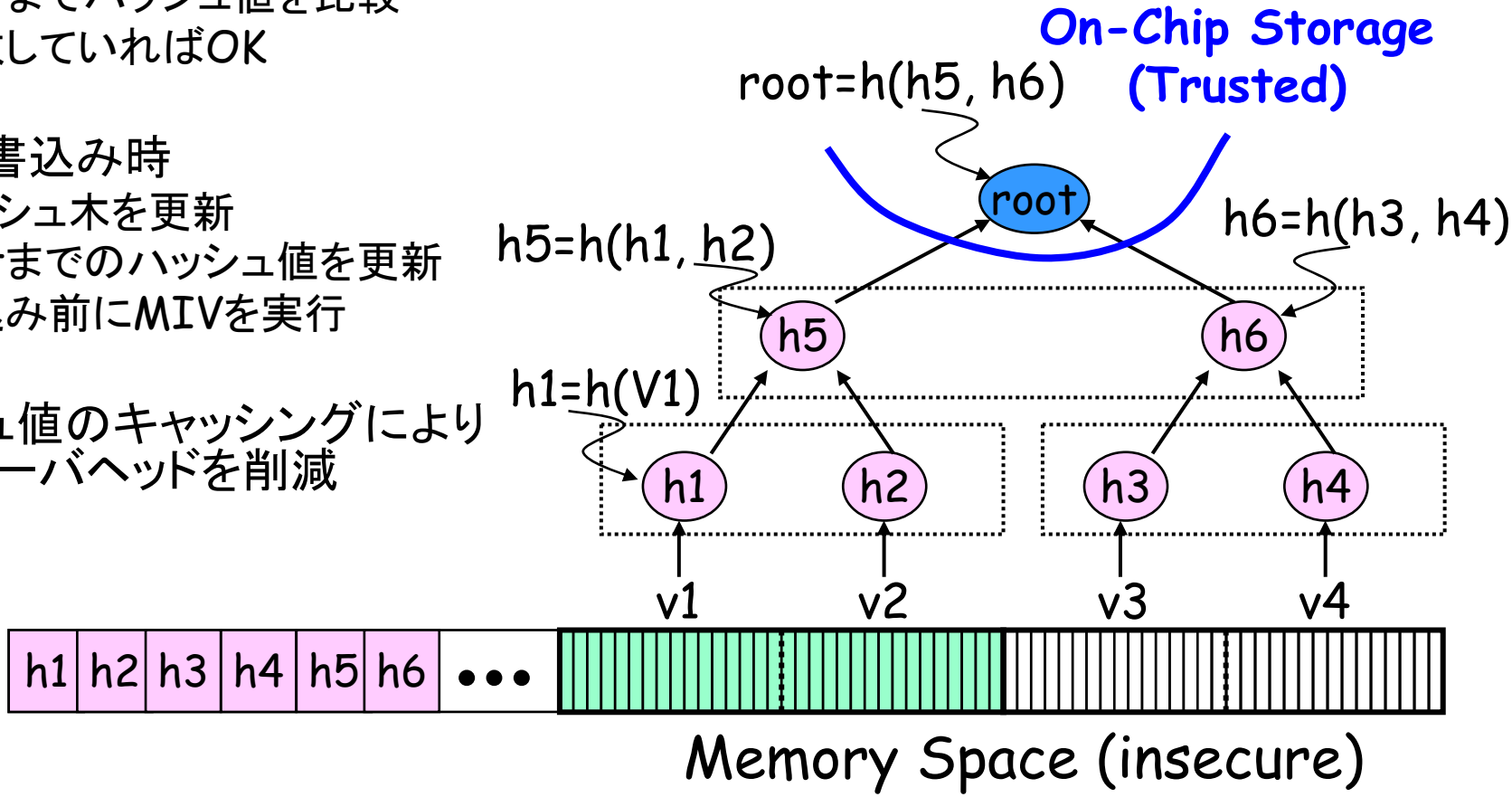
メモリ整合性検証

- M(t1)＝M(t2):
  - メモリ整合性が保たれている⇒改ざんされていない
- M(t1)≠M(t2)
  - 改ざん発生！



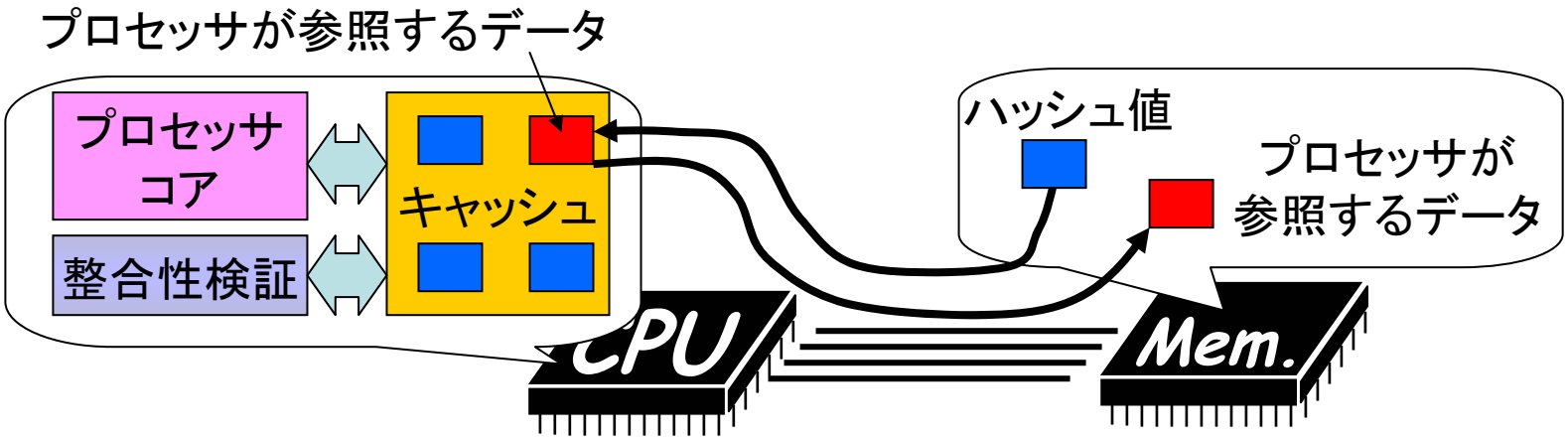
ハッシュ木によりメモリ・イメージのダイジェストを保存する！

- 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



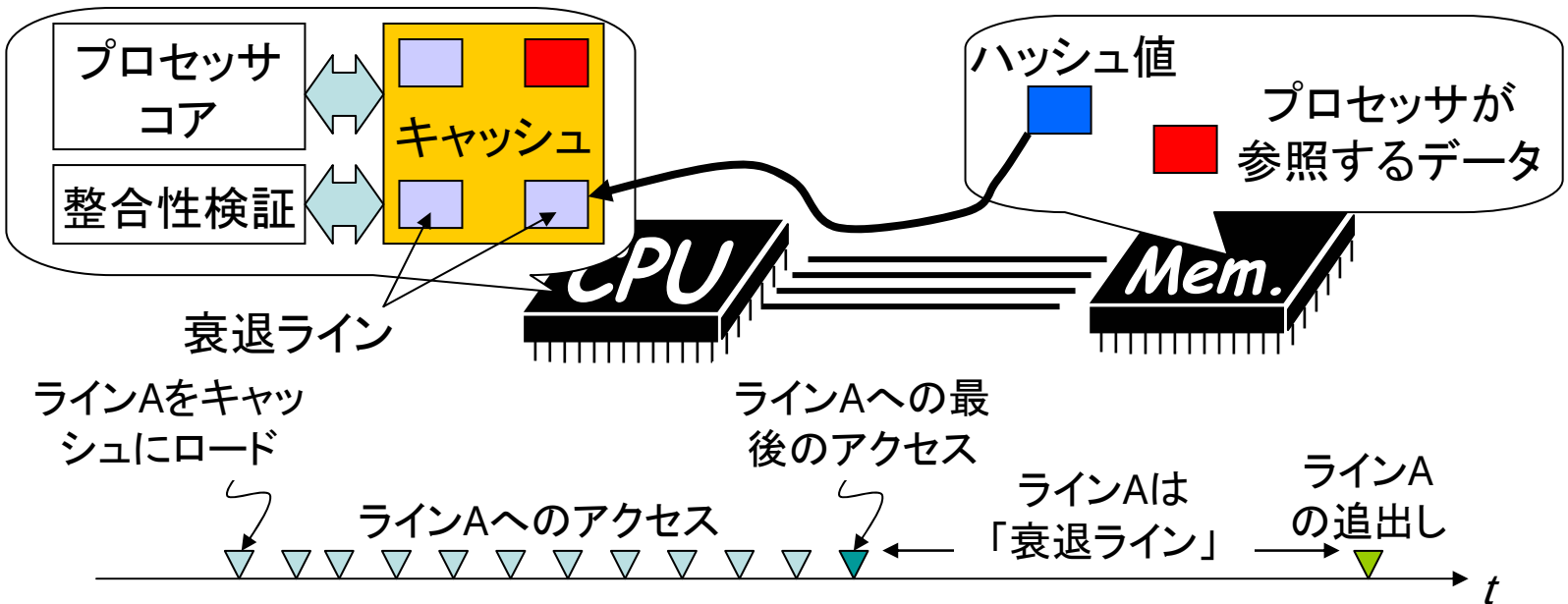
メモリ整合性検証により プロセッサ性能が低下する！

- 理由：プログラム実行処理とメモリ整合性検証処理が様々な「資源」を共有する！
- 具体的には・・
  - オンチップ・キャッシュ（⇒ミス率の増加）
  - オフチップ・メモリバス（⇒メモリバンド幅の浪費）



キャッシュの衰退ラインを利用した性能オーバヘッドの抑制

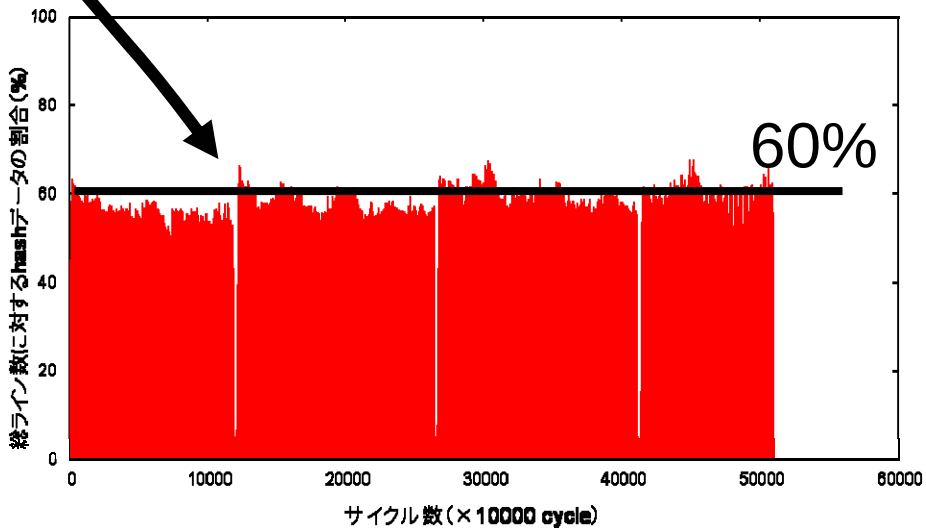
- キャッシュの衰退ラインを活用
  - 衰退ライン⇒プロセッサから参照されない（使用済みの）データを保持しているデータブロック
  - 衰退ライン（使用済みライン）にハッシュ値を格納
    - ハッシュ値のキャッシング時，プログラム実行に必要なデータを追い出さないようにする



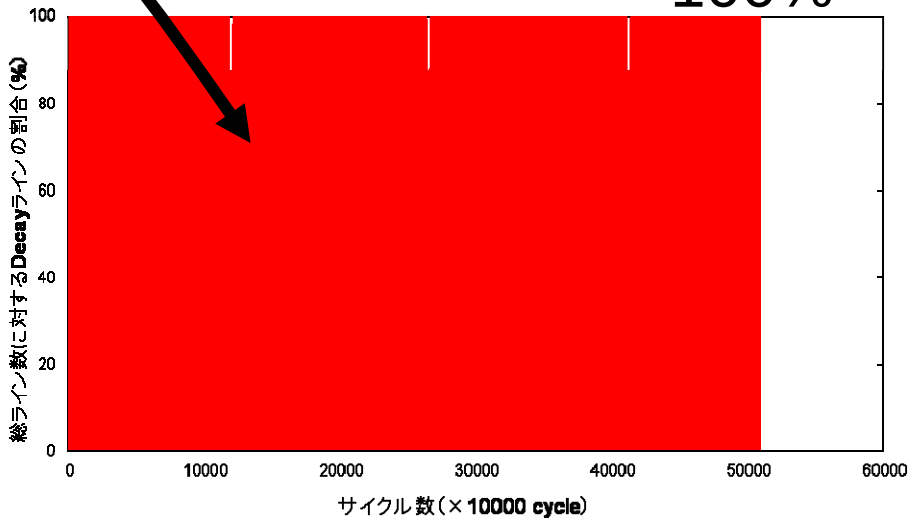
衰退ラインはどの程度存在するのか？

	177.mesa	167.gzip	255.vortex
キャッシュ中に格納すべきハッシュ値の割合(%)	73.2	34.7	56.2
キャッシュ中に存在する衰退ラインの割合(%)	99.1	98.3	98.4

L2キャッシュサイズ：128KB

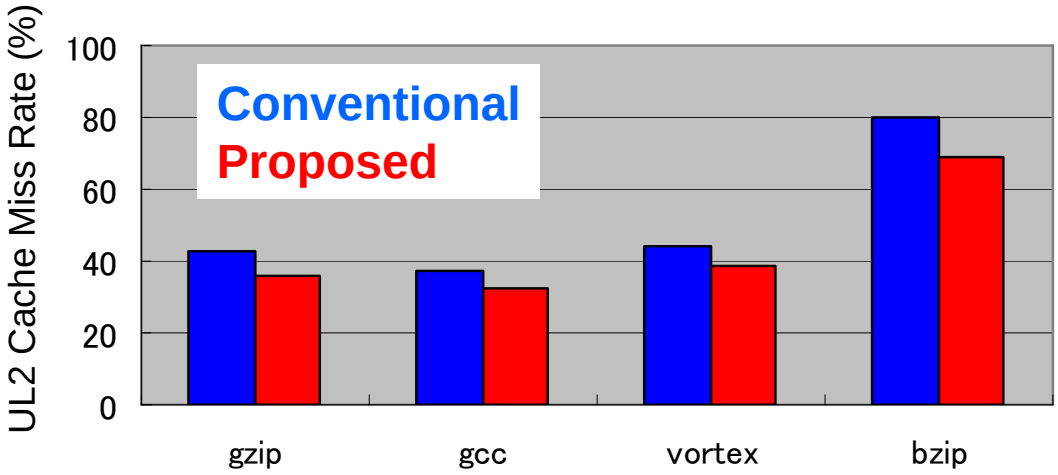


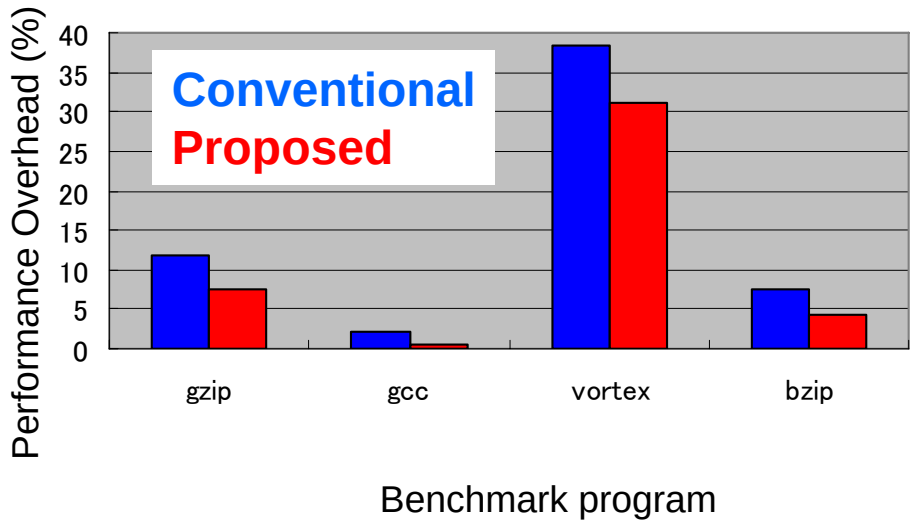
100%



255.vortex

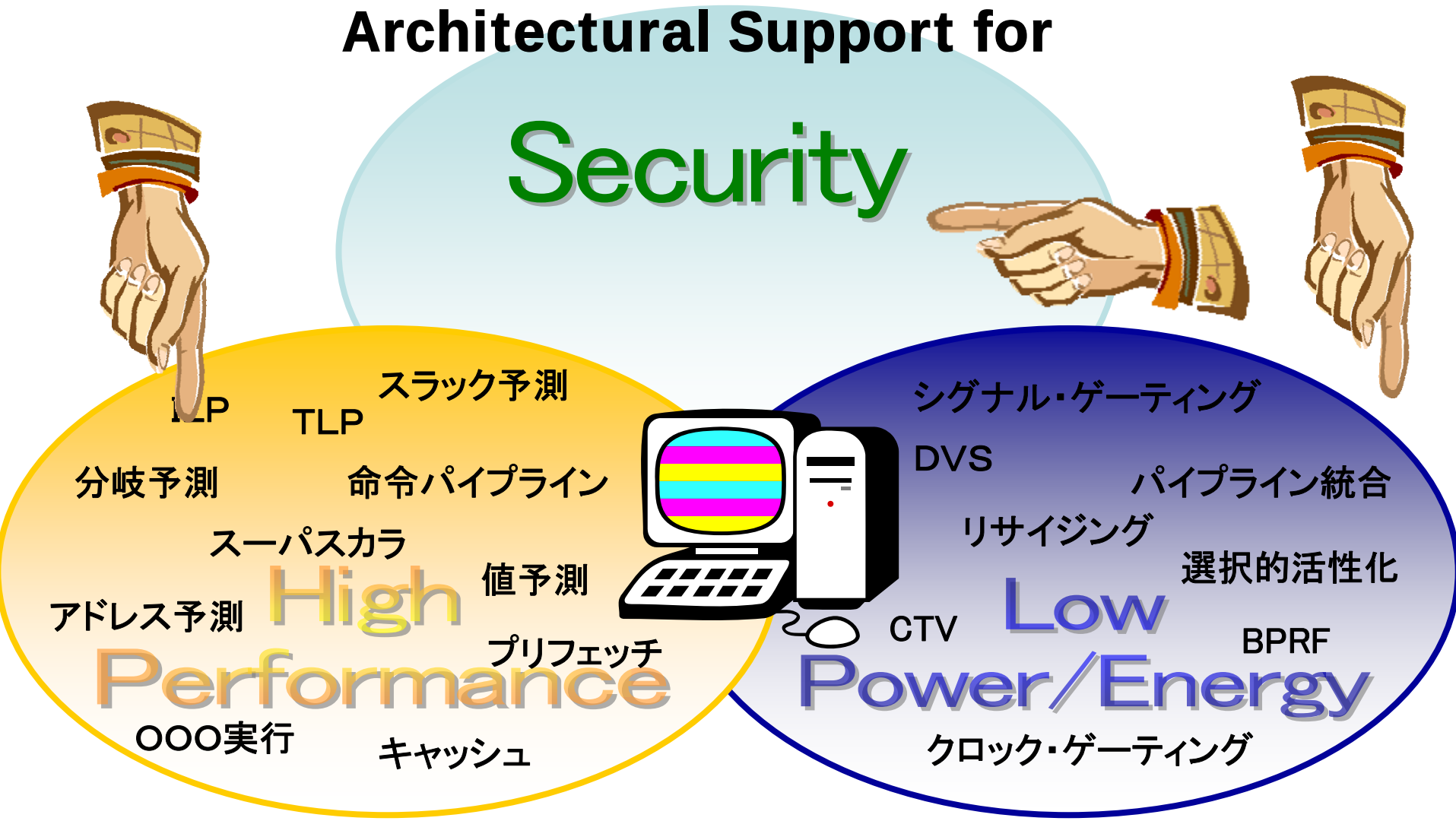
プロセッサ性能オーバヘッドの削減





4-way OOO Superscalar Processor
128KB 4-way set-associative UL2 Cache
キャッシュミスによる影響のみ考慮

本研究で目指すところ・・・


Architectural Support for
Security
スラック予測
TLP
TLP
分岐予測
命令パイプライン
スーパスカラ
値予測
アドレス予測
High
Performance
プリフェッチ
OOO実行
キャッシュ
シグナル・ゲーティング
DVS
パイプライン統合
リサイジング
選択的活性化
CTV
Low
Power/Energy
BPRF
クロック・ゲーティング