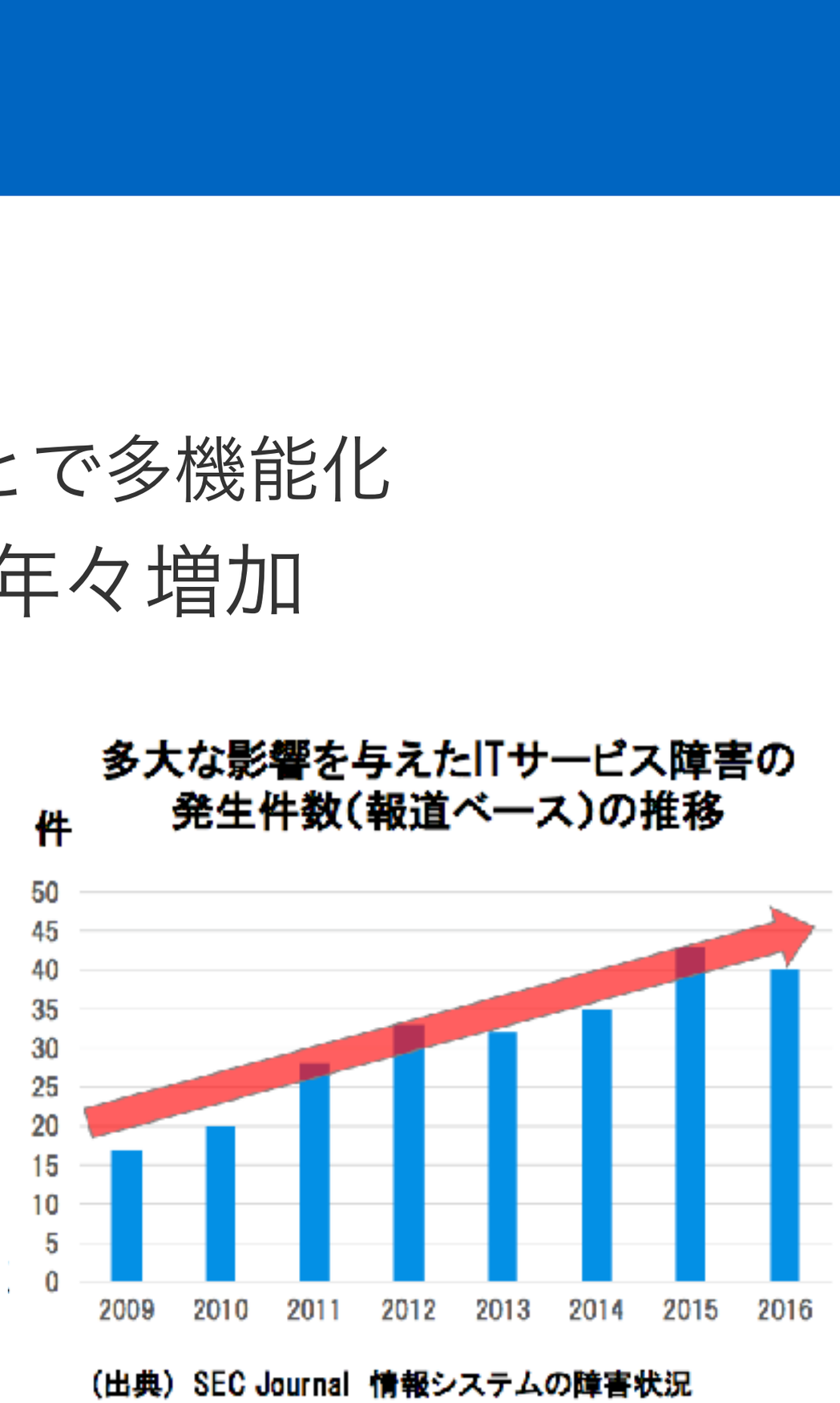
GPUを用いたOSレベルでの障害検知

九州工業大学
尾崎 雄一, 山本 裕明, 光来 健一

システム障害

- 情報システムの複雑化
  - 多様な技術を取り入れることで多機能化
- システム障害の発生件数が年々増加
  - ここ8年で倍増
- 大きな損失が生じる
  - サービス提供者
  - システム管理者
  - ユーザ



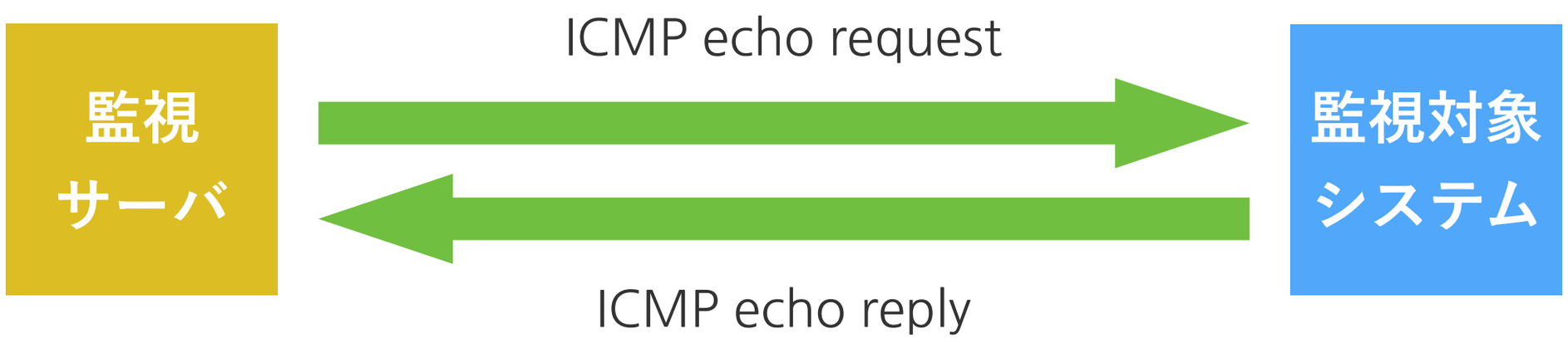
(出典) SEC Journal 情報システムの障害状況

障害検知の重要性

- 損失を軽減するために障害から素早く復旧し、サービス提供を再開することが必要
  - 障害をできるだけ早く検知することが重要
    - 障害発生前に予兆を検知することが望ましい
  - 障害をできるだけ正確に検知することも重要
    - 障害の誤検知は損失につながる

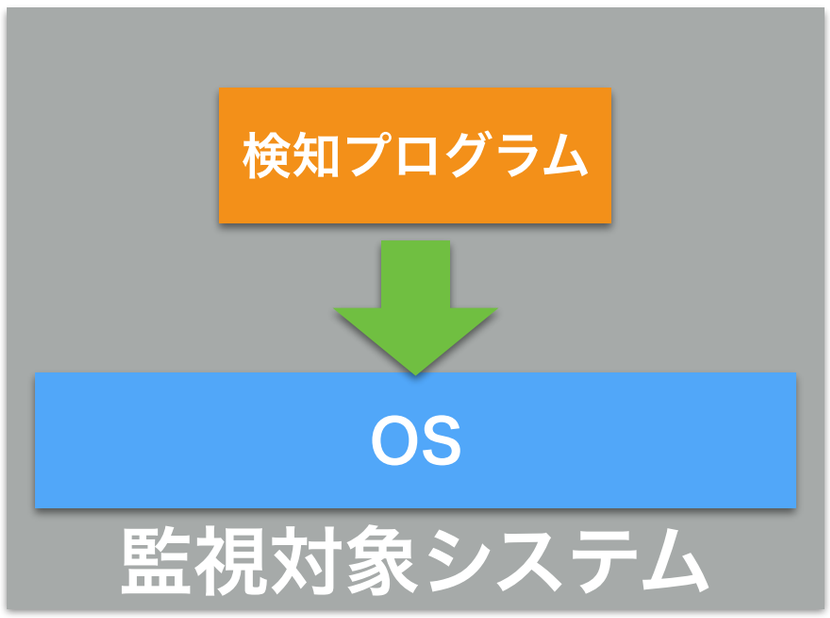
従来のシステム監視方法 (外部)

- システム外部からネットワーク経由で監視
  - システムの状態に依存せずに障害検知できる
  - 障害発生時に詳細な情報を取得できない
- システムがVM内で動作する場合は仮想ハードウェアの状態も取得可能
  - それ以上の詳細な情報は得られない



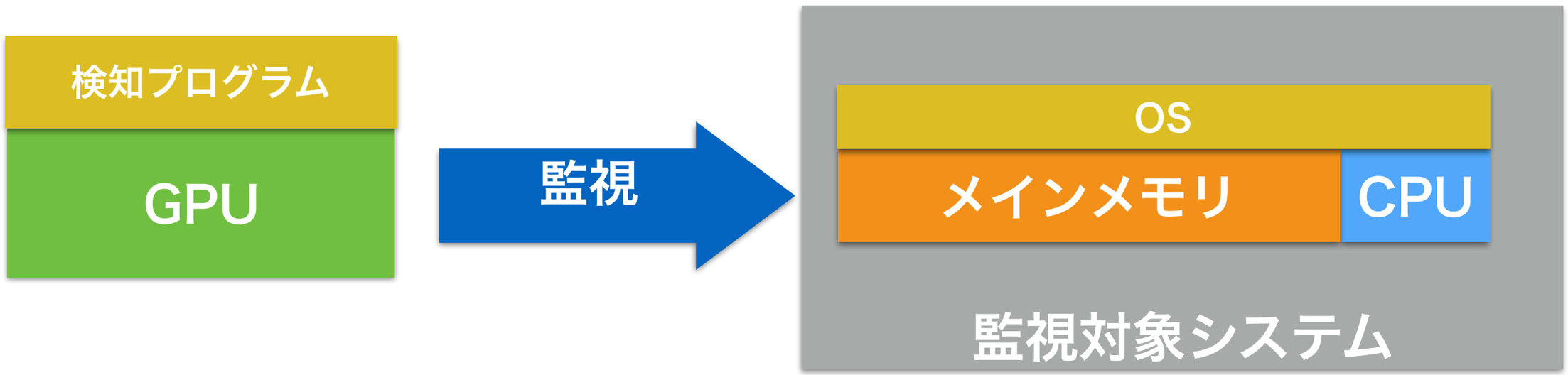
従来のシステム監視方法 (内部)

- システム内部で検知プログラムを実行して内部状態を直接監視
  - システムの詳細な情報を利用した監視が可能
    - 例: CPU使用率の高いプロセスを特定
  - 障害の影響を受けやすい
    - OSの障害による機能停止
    - メモリ不足による強制終了
- システム全体の性能低下の可能性



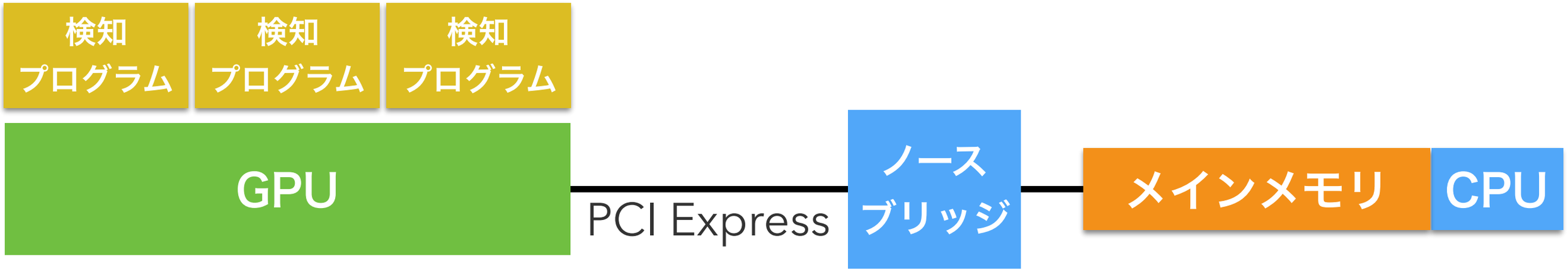
提案: GPUSentinel

- システムが動作するホストのGPU上で障害を検知
  - 検知プログラムがGPUを占有して自律的に動作
    - 障害が発生する前のシステム起動時に実行開始
  - システムのメインメモリ上のデータを解析
  - OSレベルの詳細な情報を利用した精度の高い障害検知



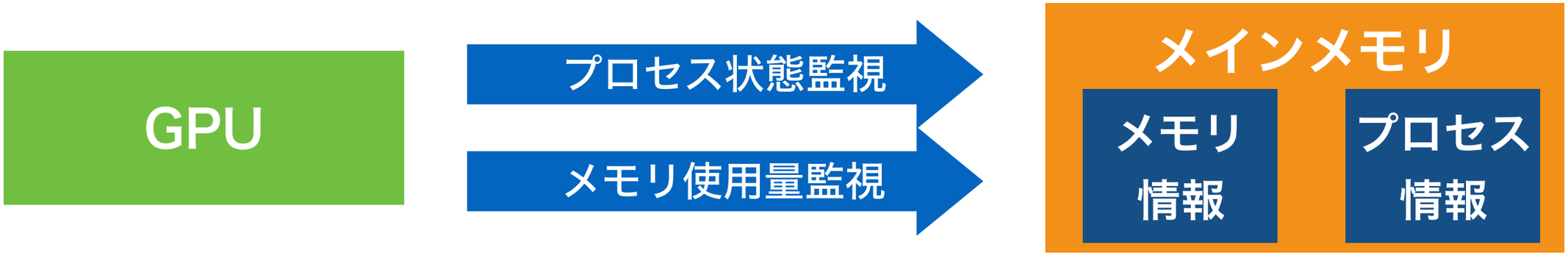
GPUを用いる利点

- GPUはシステム障害の影響を受けにくい
  - システムが動作するCPUやメインメモリから独立
- システムへの影響を抑えて様々な監視が可能
  - 同時に多数の監視が行え、それぞれの監視も並列実行
- CPU性能に影響を与えない
  - メインメモリやPCI Expressの帯域は消費



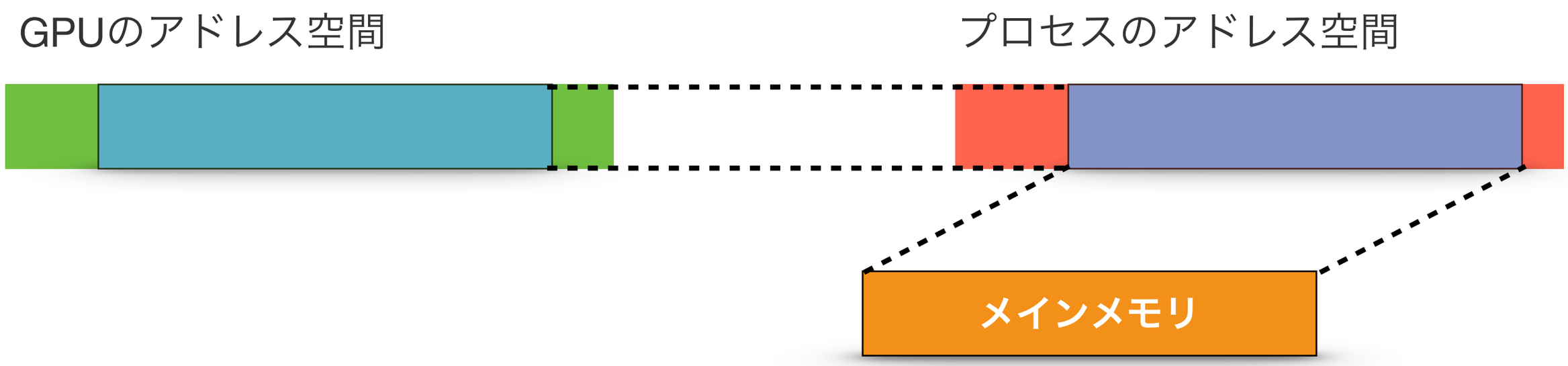
検知可能な障害

- メインメモリ上のデータから見つけられる障害
  - 例: システムリソースの枯渇
    - 利用可能なメモリ量の情報からメモリ不足を検知
    - プロセス数の増加から異常を検知
- それ以外のハードウェア障害の検知は難しい
  - GPUはメインメモリ以外にはアクセスできない



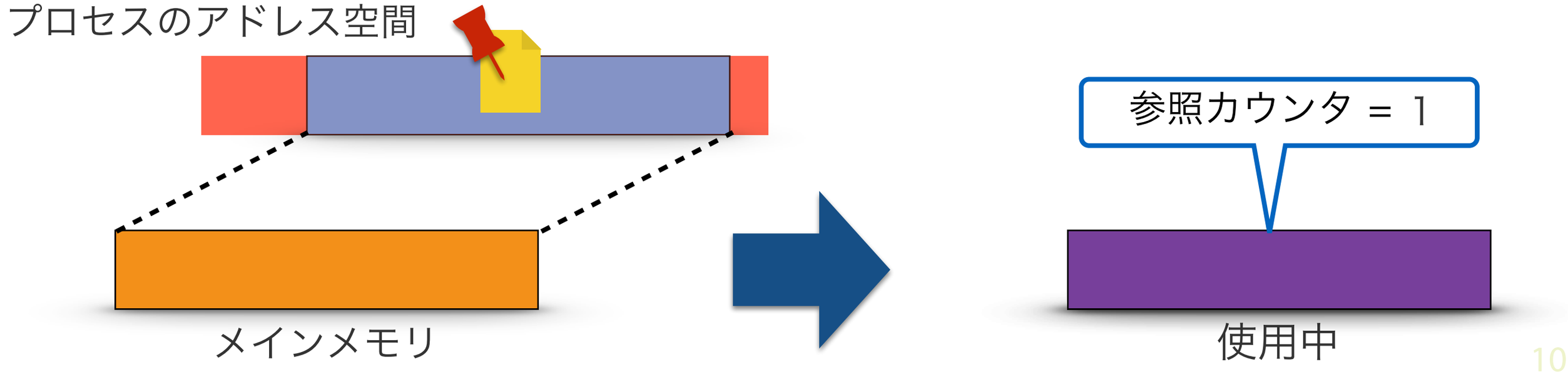
GPUからメインメモリへのアクセス

- CUDAのマップトメモリ機能を利用
  - メインメモリ全体をプロセスのアドレス空間にマップ
  - それをGPUのアドレス空間にマップ
    - GPUからアクセスすると透過的にDMA転送
- システムに障害が発生してもメインメモリを参照可能
  - 障害が発生する前にマップしておく



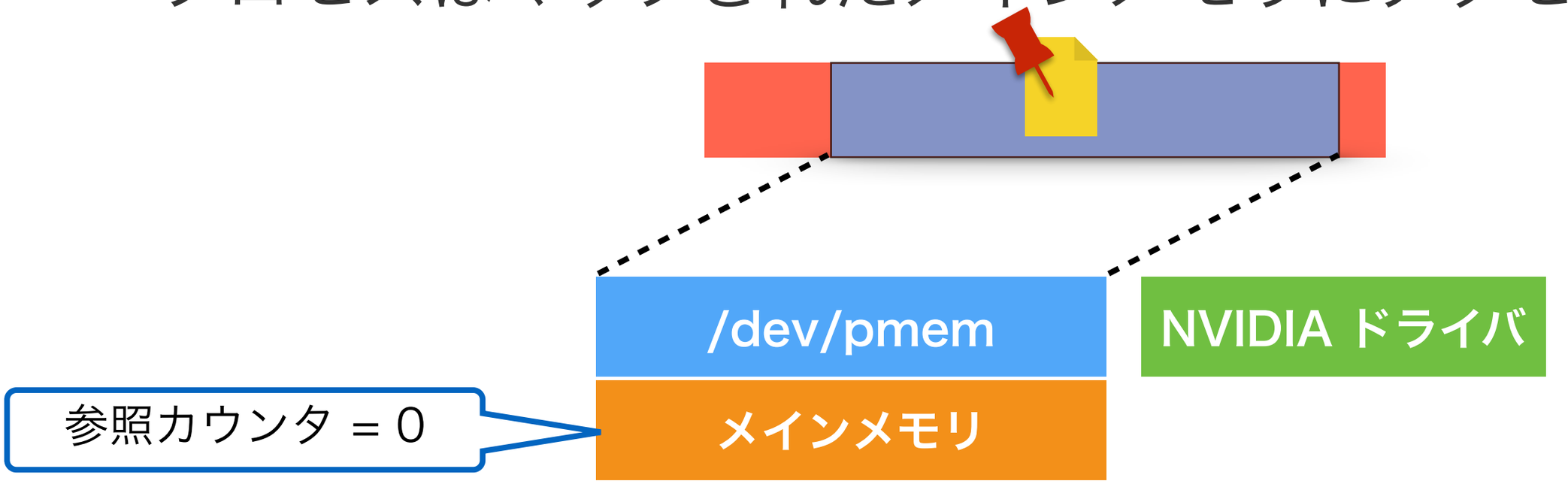
マップトメモリを用いる際の問題

- メインメモリ全体をGPUにマップするとシステムの空きメモリがなくなる
  - CUDAがプロセスにマップしたメインメモリをピン留めする
    - ページアウトされないようにするため
  - ピン留めするとメモリは使用中になる



GPUSentinelのメモリ管理

- メインメモリ全体をGPUにマップ可能にする機構
  - /dev/pmemという特殊なデバイスファイルを用意
    - ピン留めの際にページの参照カウンタを増やさない
  - NVIDIAドライバと連携して正常にピン留めを解除
    - ピン留めを解除する際に参照カウンタを減らさない
  - プロセスはマップされたメインメモリにアクセス不可

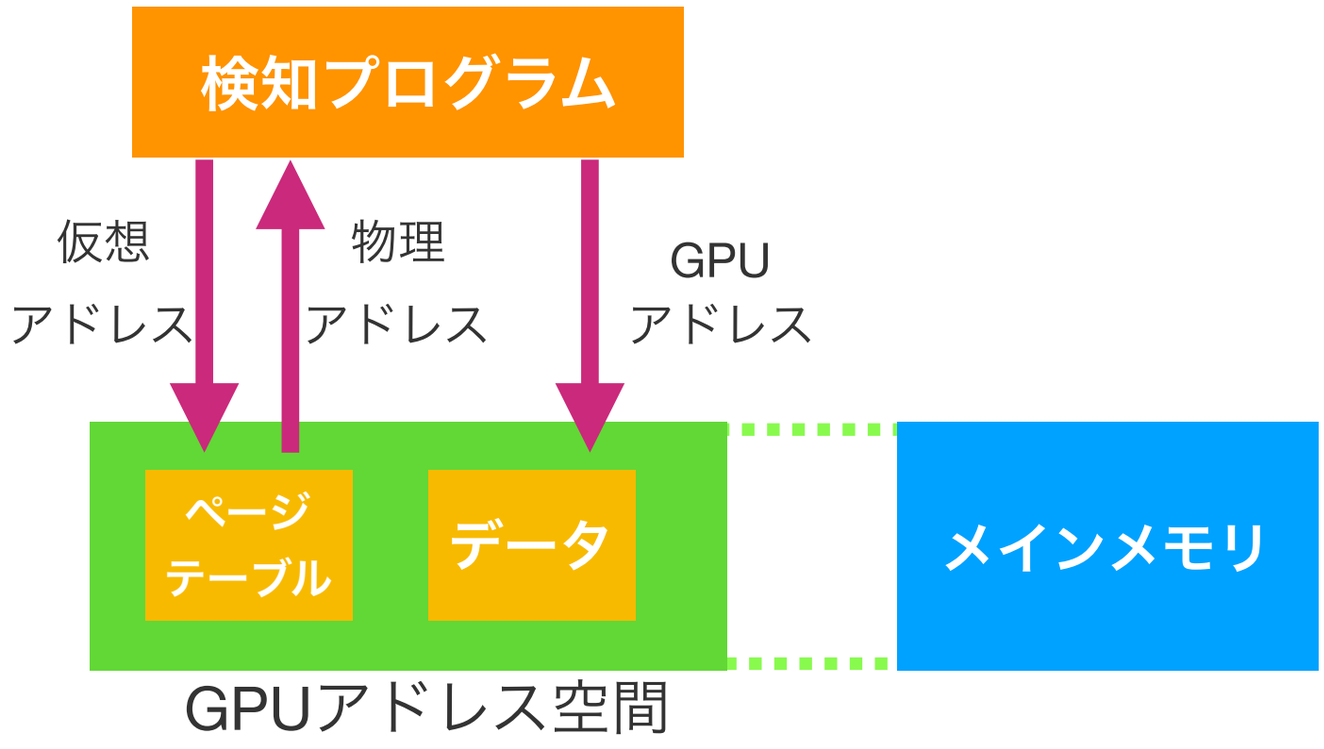


メインメモリ全体のマップ

- マップトメモリ機構の制限を回避するためにsysinfoシステムコールを偽装
  - CUDAではメインメモリより小さな領域しかマップできない
- LD_PRELOADでsysinfoシステムコールをフック
  - メインメモリのサイズとして少し大きい値を返す

アドレス変換の必要性

- 検知プログラムはOSデータの仮想アドレスをGPUアドレスに変換する必要
  - OSのページテーブルを用いて物理アドレスに変換
  - それをGPUアドレスに変換
- 検知プログラムに変換処理を記述するのは冗長かつ煩雑



LLViewによる自動アドレス変換

- 透過的にアドレス変換を行うためのLLViewを開発
  - LLVMでコンパイルして生成された中間表現を変換
    - load命令の直前にアドレス変換関数の呼び出しを挿入
  - カーネル変数を対応する仮想アドレスに置換

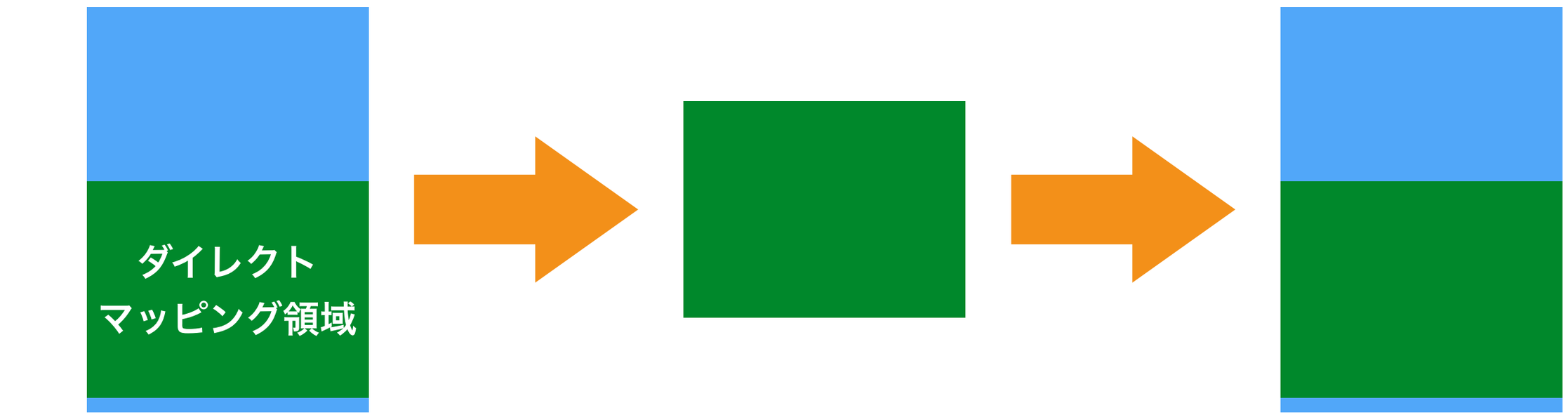
```
%1 = load i64, i64* %jiffies
%2 = udiv i64 %1, 250
```

```
%1 = bitcast i64* %0xffffffff82011500 to i8*
%2 = call i8* @g_map(i8* %1)
%3 = bitcast i8* %2 to i64*
%4 = load i64, i64* %3
%5 = udiv i64 %4, 250
```

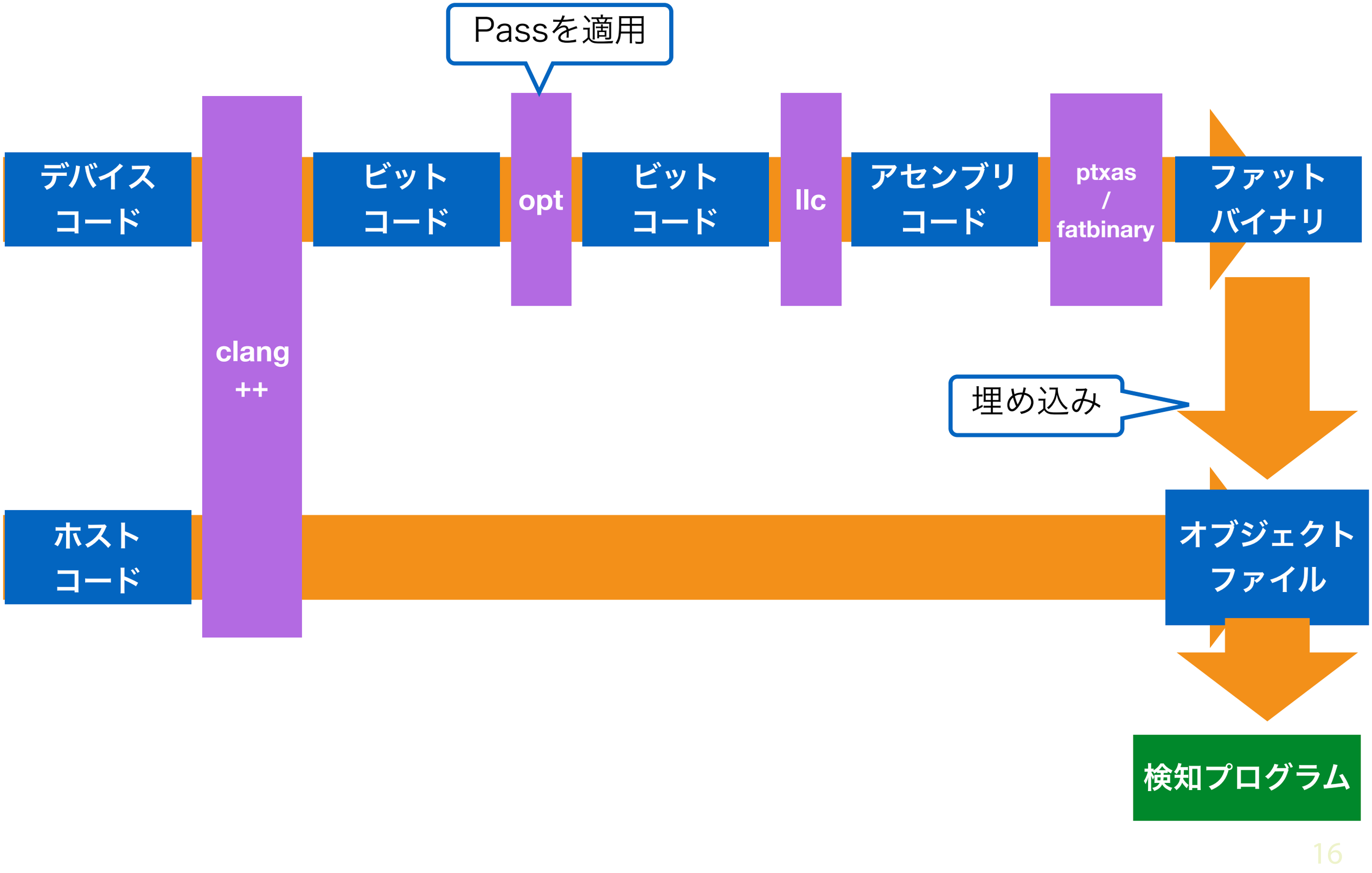
アドレス変換関数

アドレス変換関数

- ページテーブルを参照して物理アドレスに変換
  - ダイレクトマッピング領域とカーネルテキスト領域については最適化
    - 仮想アドレスから領域の先頭アドレスを引く
- 単純な計算によりGPUアドレスに変換
  - メインメモリがマップされた領域のアドレスを足す



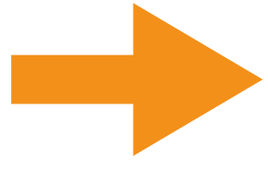
コンパイルの流れ


Passを適用
デバイス
コード
clang
++
ビット
コード
opt
ビット
コード
llc
アセンブリ
コード
ptxas
/
fatbinary
ファット
バイナリ
埋め込み
ホスト
コード
オブジェクト
ファイル
検知プログラム

Linuxヘッダファイルの利用

- Linuxカーネルのヘッダファイルを使って検知プログラムを記述
  - CUDAプログラムとしてC++でコンパイルする必要
  - 変換マクロを提供
    - C++のキーワードと衝突する変数名を置換
    - 標準Cライブラリとの名前の衝突を回避
- C++で必要になるキャストを追加

```
new
struct timeval
(void *)addr + …
```



```
_new
struct _timeval
(char *)addr + …
```

実験

- GPUSentinelの有用性を調べる実験を行った
  - OSのプロセスリストをたどってプロセス情報を取得する検知プログラムを作成
  - GPU上での検知プログラムの動作を確認
- システム情報の取得にかかる時間を測定

GPU	GeForce GT610
メインメモリ	16GB
Linux	4.4.67
CUDA	8.0.61
NVIDIAドライバ	375.66

動作確認

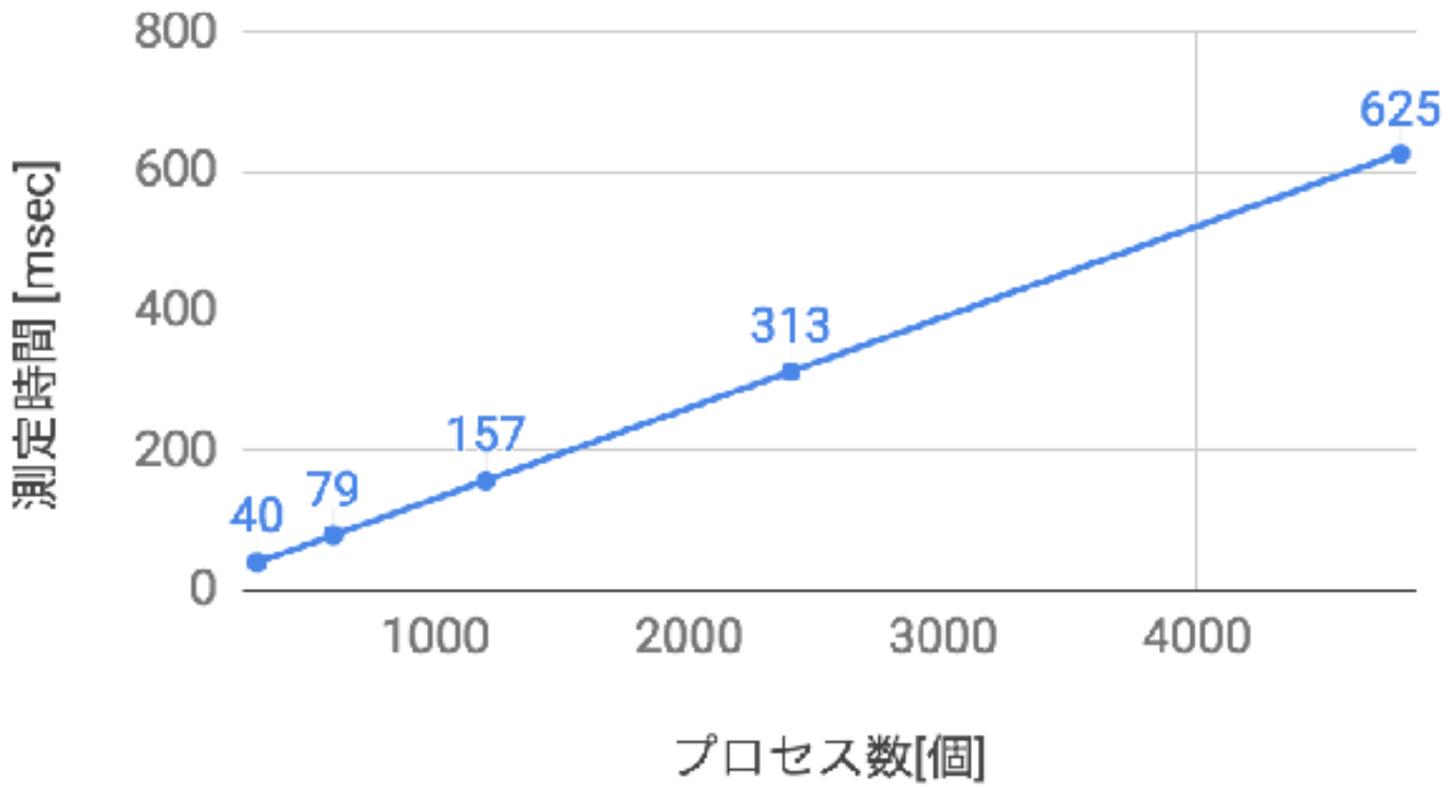
- 以下の検知プログラムをGPU上で実行
  - カーネルモジュールとほぼ同じ記述
  - 正常に300個のプロセスのIDと名前が表示された

```
struct task_struct *p;

p = &init_task;

do {
    printf("pid: %d\n", p->pid);
    printf("comm: %s\n", kstr(&p->comm));

    p = list_entry(p->tasks.next,
          struct task_struct, tasks);
} while (p != &init_task);
```

```
pid: 0
comm: swapper/0
pid: 1
comm: systemd
pid: 2
comm: kthreadd
pid: 3
comm: ksoftirqd/0
```

監視性能

- システム上で実行中のプロセス数を変えながら情報取得にかかる時間を測定
  - プロセス数に比例して取得にかかる時間が増加
    - プロセス数が多いと障害検知に時間がかかる可能性
    - 検知プログラムを並列化することで改善可能



関連研究

- カーネル間メモリ監視による障害検知 [松下ら'17]
  - 1台で2つのOSを動作させて監視対象OSを監視
  - 監視OS用にCPUとメモリを割り当てる必要がある
- Copilot [Petroni et al.'04]
  - 専用のPCIカードを用いてカーネルメモリを監視
  - 専用ハードウェアのコストが高い
- SPE Observer [Kourai et al.'12]
  - Cell/B.E.の隔離されたSPEを用いてOSを監視
  - Cell/B.E.プロセッサは普及していない

まとめ

- GPU上で障害検知を行うGPUSentinelを提案
  - メインメモリをマップすることでOSデータを監視
  - 自動アドレス変換を行うLLViewを開発
  - プロセス情報が実用的な時間で取得できることを確認
- 今後の課題
  - 実際に障害検知を行うプログラムの作成
  - 検知プログラムの並列化による高速化
  - 検知した障害を外部に通知する機構の開発