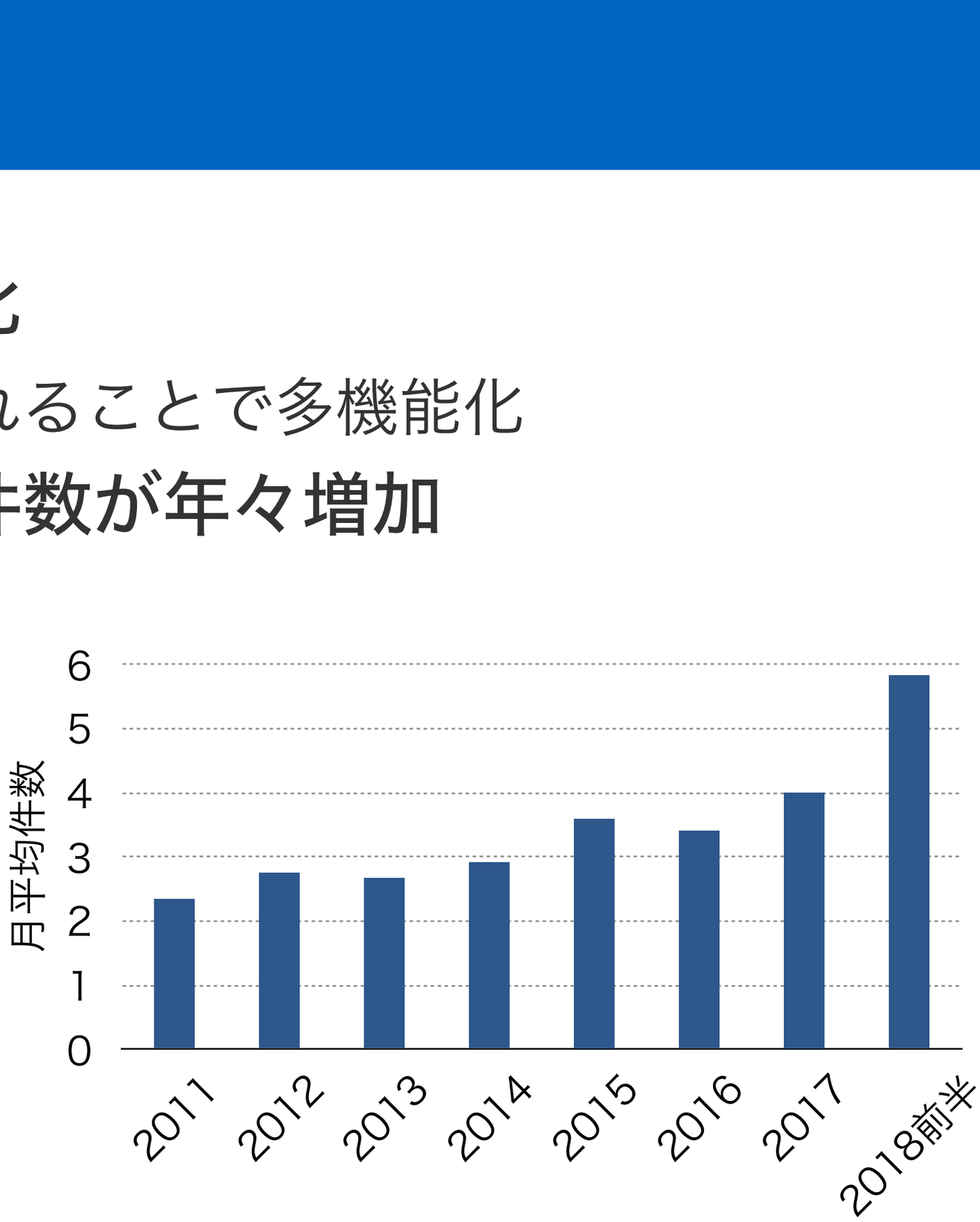
# GPUとLLVMを用いたOSレベルでの障害検知機構

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



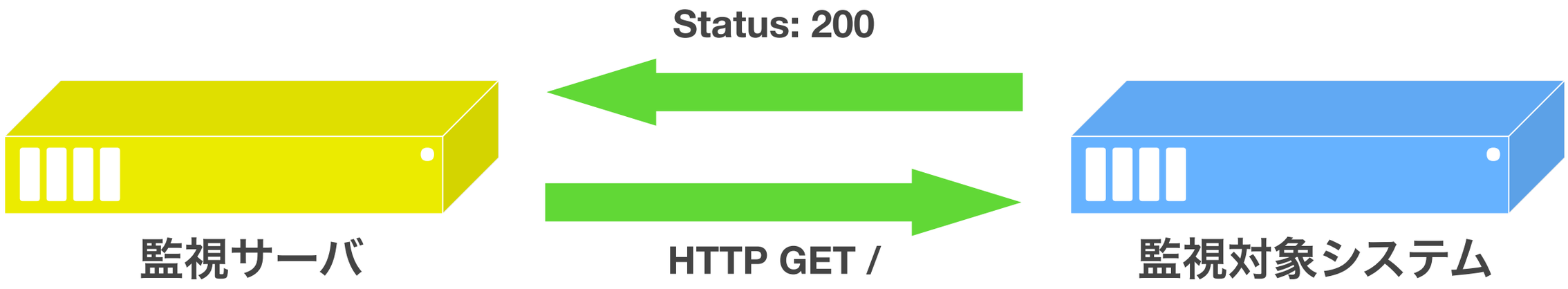

(出典) IPA: 情報システムの障害状況 2018年前半データ

# 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

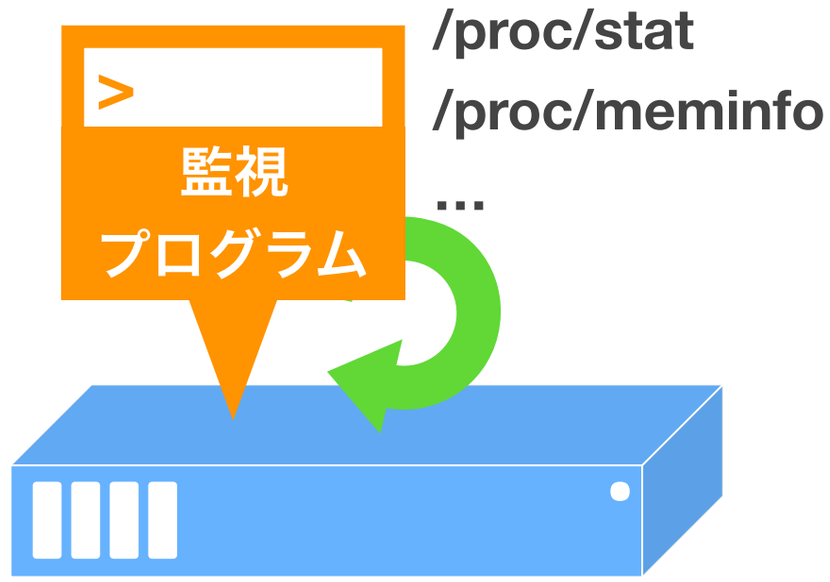
# 従来のシステム監視方法 (外部)

- システム外部からネットワーク経由で監視
  - ✓監視対象システムをブラックボックスとして障害検知できる
    - pingを用いたホストの死活監視
    - ネットワークサービスに定期的に接続
  - ×障害発生時に詳細な情報を取得できない
    - 監視対象システムの内部状態は分からない



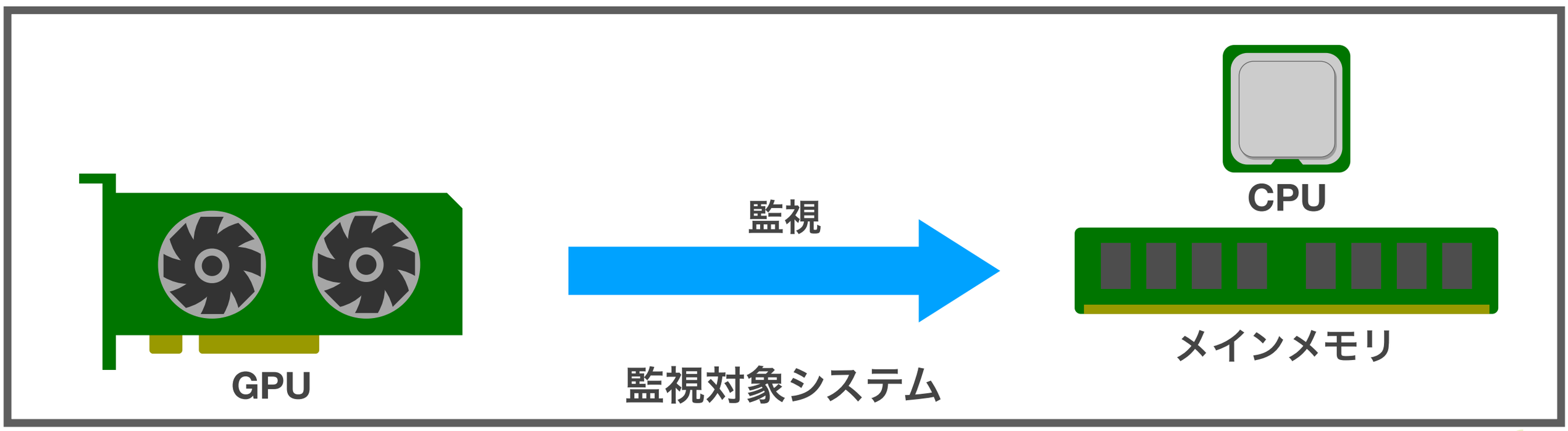
# 従来のシステム監視方法 (内部)

- 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

- 監視対象ホストの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
  - ✓メインメモリ上のOS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

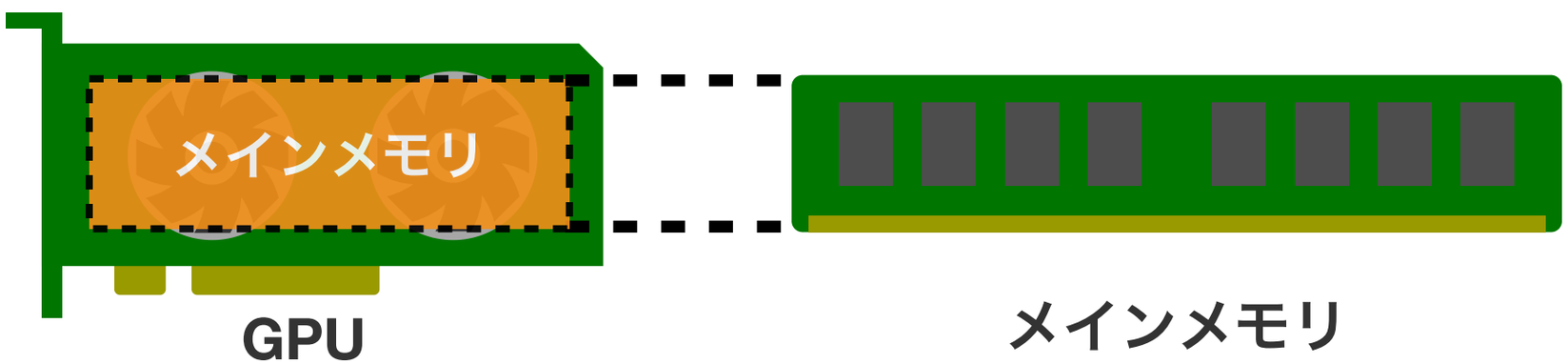


# GPUを用いる利点

- **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
- **システムの性能低下を抑えることができる**
  - CPU性能に影響を与えない
- 同時に様々な監視が可能
  - 多数の演算コアを持っている

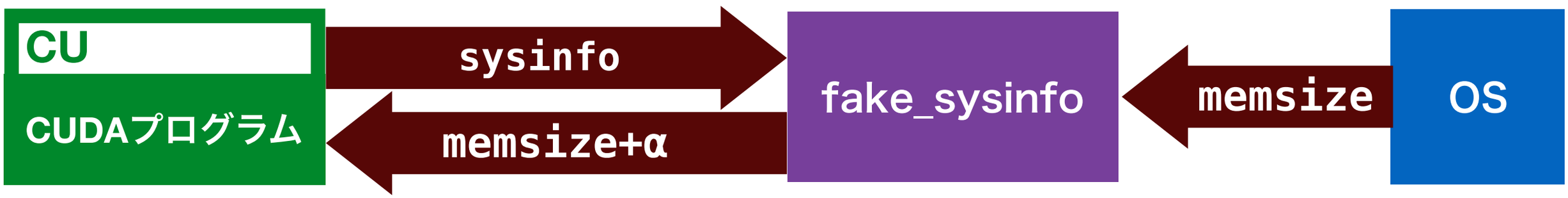
# GPUからメインメモリへのアクセス

- **CUDAのマップトメモリ機能を利用**
  - 事前にメインメモリ全体をGPUにマッピング
  - 障害発生時でもメインメモリを参照可能
- **メインメモリ全体をマップするとシステムの空きメモリがなくなる**
  - マップしたメモリがピン留されて使用中になるため



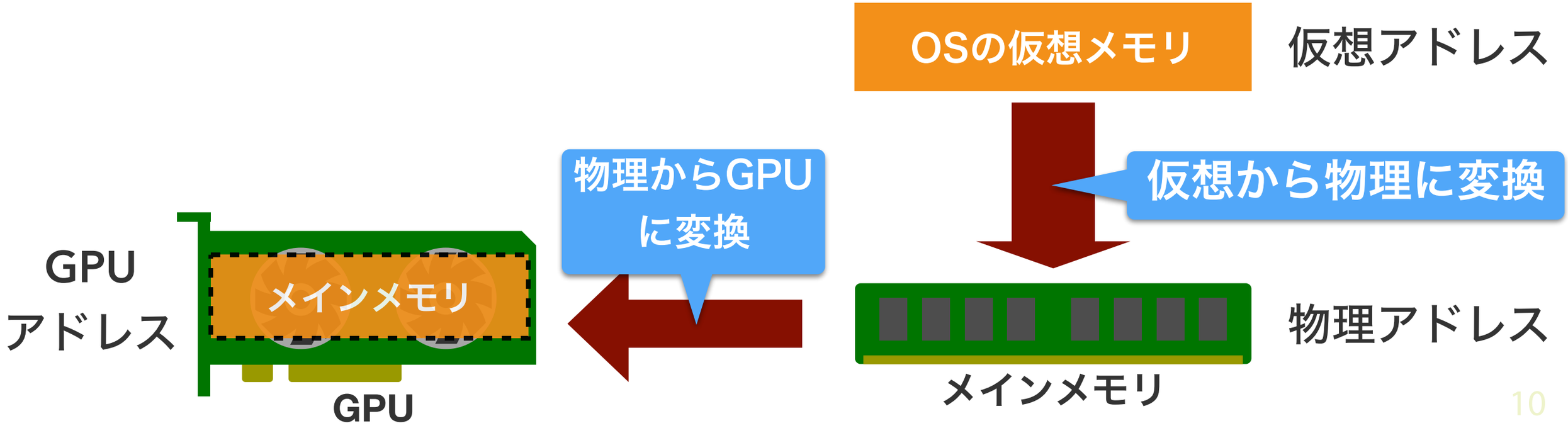
# GPUSentinelのメモリ管理機構

- メインメモリ全体をGPUにマップ可能にするメモリ管理機構を提供
  - 提供される特殊なデバイスファイルをマップ
  - メモリをピン留めしても参照カウンタを増減させない
  - CUDAの制限を回避するためにメモリサイズを偽装



# アドレス変換の必要性

- **GPUからメインメモリ上のOSデータを参照するにはアドレス変換が必要**
  - OSのデータはOSの仮想メモリにあり、仮想アドレスで管理されている
  - GPUはGPUメモリにGPUアドレスでアクセスする



# 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**
  - clang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
    - OSの大域変数を対応するアドレスに置換

```
%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

# GPU上の検知プログラム

- **OSのソースコードを最大限に利用して作成**
  - ✓OSの構造体や大域変数を利用するCプログラム
    - CUDAプログラムをC言語としてコンパイルできるようにclangコンパイラを修正

```
struct sysinfo system_info;
unsigned long freeram, freeswap;

si_meminfo(&system_info);
si_swapinfo(&system_info);

freeram = system_info.freeram;
freeswap = system_info.freeswap;
```

# 検知プログラムの例(1/2)

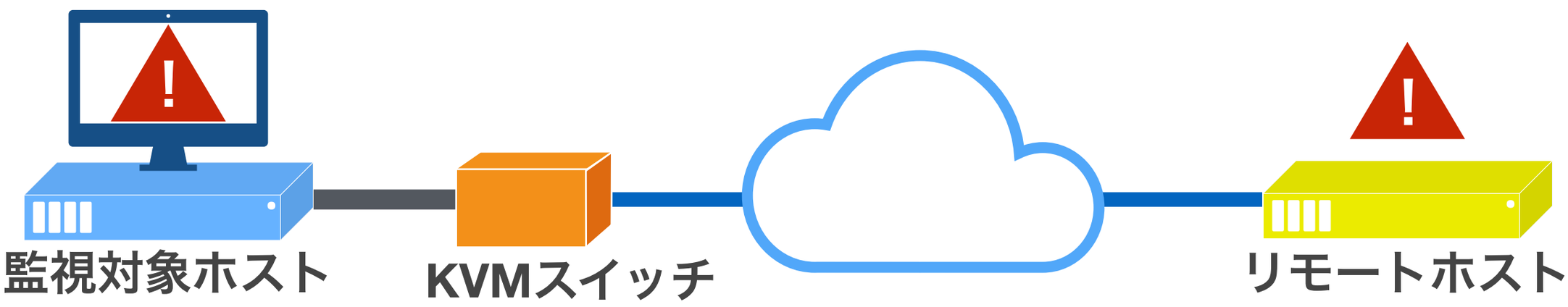
- **デッドロックによるOSのハングアップの検知**
  - スピンロックによる全CPUのデッドロック
  - CPU時間とコンテキストスイッチ回数を監視
    - 全CPUのシステム時間が**95%**以上かつユーザ時間が**5%**未満で検知
      - スピンロック待ちはカーネル空間で行われる
      - スピンロック中は他のタスクを実行しない
    - または、コンテキストスイッチ回数が**0**で検知

# 検知プログラムの例(2/2)

- **CPUの高負荷状態の検知**
  - CPU使用率を監視
    - 全CPUの使用率が**90%**以上の状態が**5秒**以上経過で検知
  - プロセスごとの使用率を計算して原因を特定
- **メモリの枯渇の検知**
  - メインメモリとスワップの空き容量を監視
    - 両方とも**30%**以下で検知
  - 各プロセスのOOMスコアを計算して原因を特定

# 障害発生の通知

- リアルタイム出力機構
  - ✓リングバッファを用いてGPUからホストプロセスにリアルタイムにデータを送信
  - ×障害によってOSが停止すると利用できなくなる
- 画面へのダイレクト出力機構
  - ✓GPUからメインメモリ上のVRAMに書き込むことで画像を出力



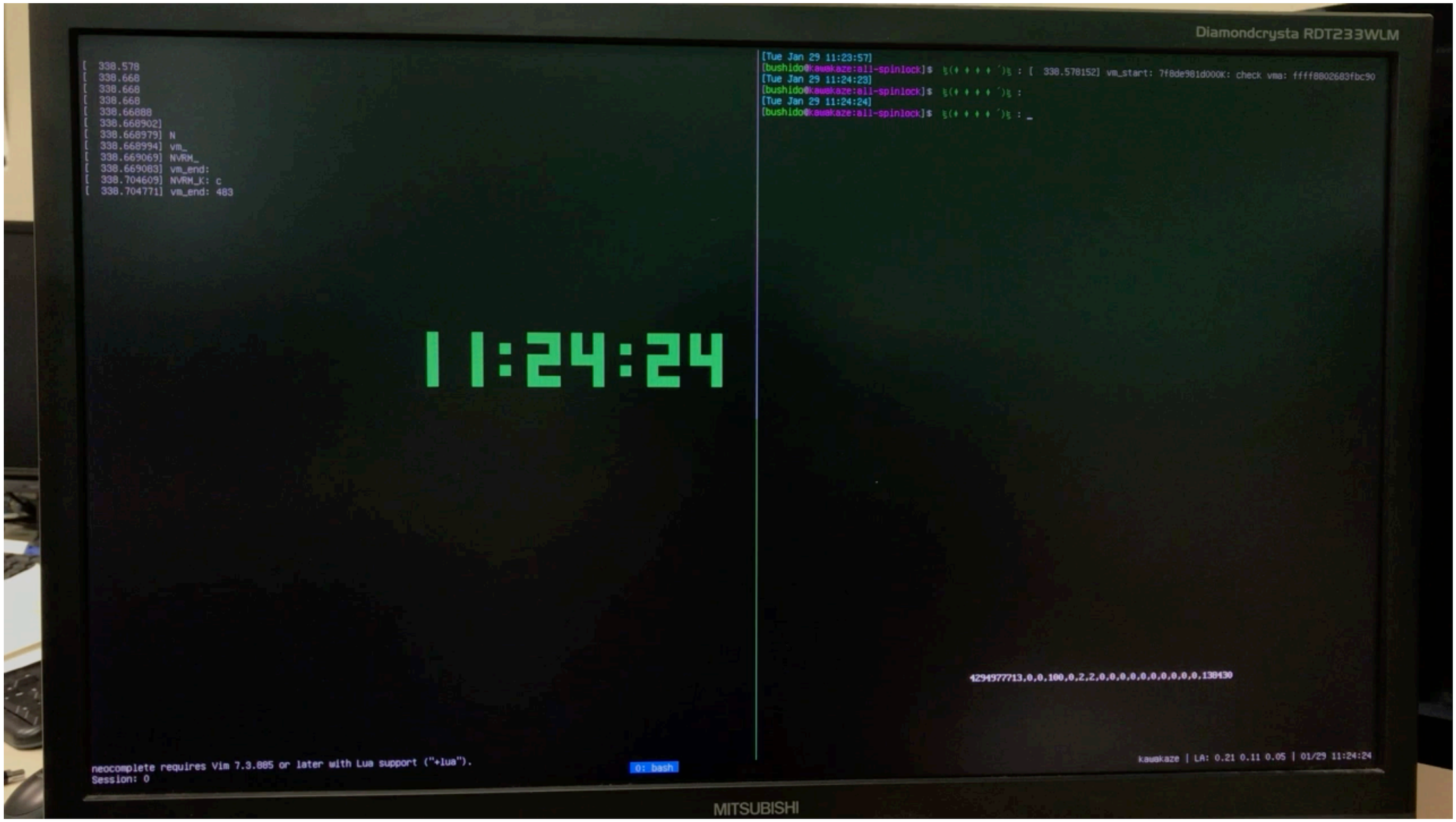
# 実験

- **GPUSentinelで障害検知が可能かどうかを調べた**
  - デッドロック、CPUの高負荷状態、メモリの枯渇
- **既存の監視システムとの比較を行った**
  - Zabbixによるリモート監視
- **GPUSentinelのオーバヘッドを調べた**
  - メインメモリ帯域への影響

| | |
|---|---|
| GPU | GeForce GTX 960 |
| メインメモリ | 8GB (DDR4-2400) |
| Linux | 4.4.67 |
| CUDA | 8.0.61 |
| NVIDIAドライバ | 375.66 |

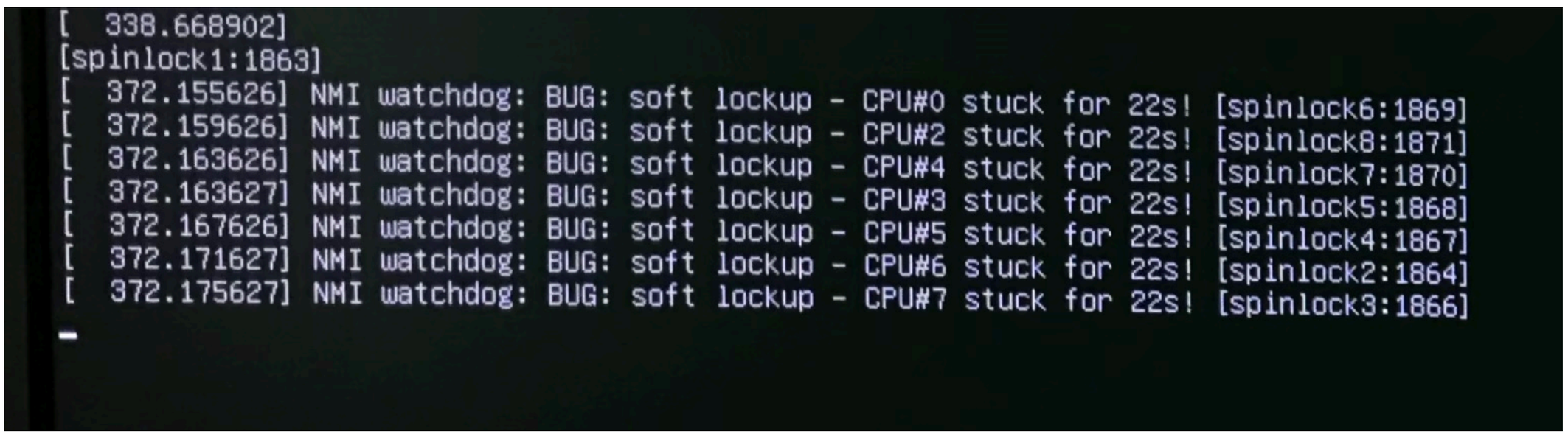
# OSがハングアップする障害の検知(1/2)

- スピンロックにより全CPUをデッドロックさせるカーネルモジュールを挿入



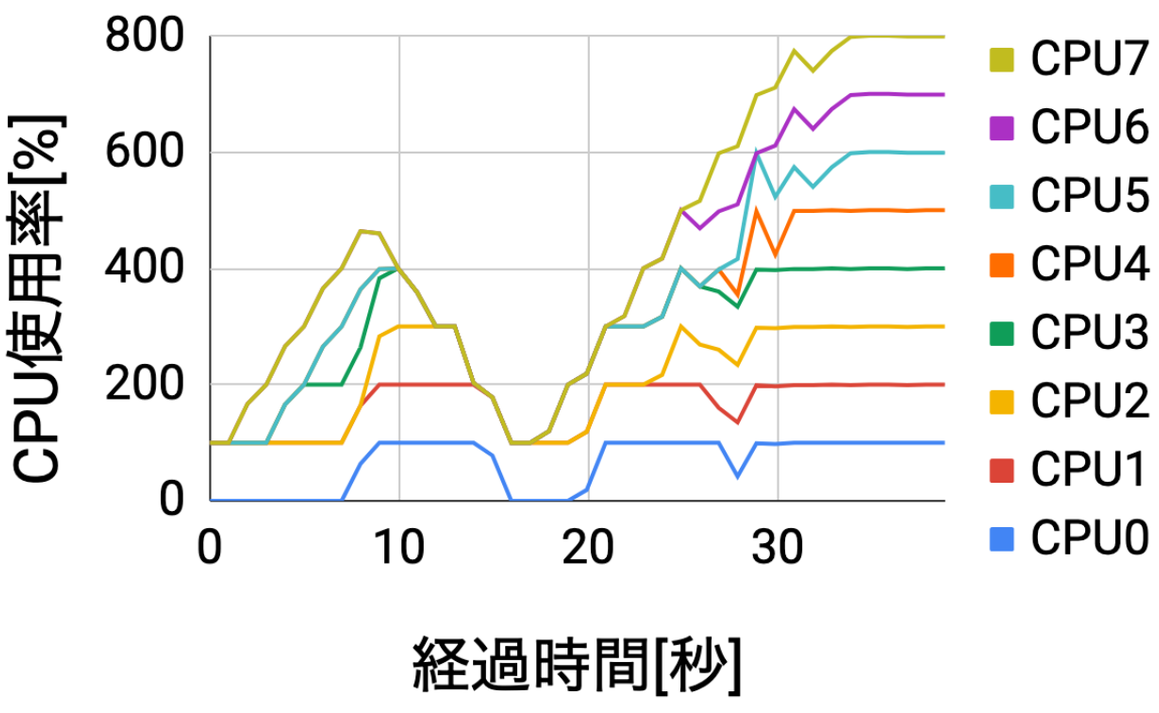
# OSがハングアップする障害の検知(2/2)

- デッドロックを1.7秒で検知
  - 障害検知を通知する画像が表示された
  - コンテキストスイッチ回数が0であった
- OSのWatchdogタイマより早く検知できた
  - 検知の数秒後にソフトロックアップ・メッセージが表示された



# CPUの高負荷状態の検知

- CPUに負荷をかける処理を並列に実行するプログラムを実行
  - ✓8プロセスで負荷をかけ始めてから4.7秒で検知
  - ✓負荷の原因となったプロセスの名前を出力
    - ただし、高負荷というだけであって異常とは限らない

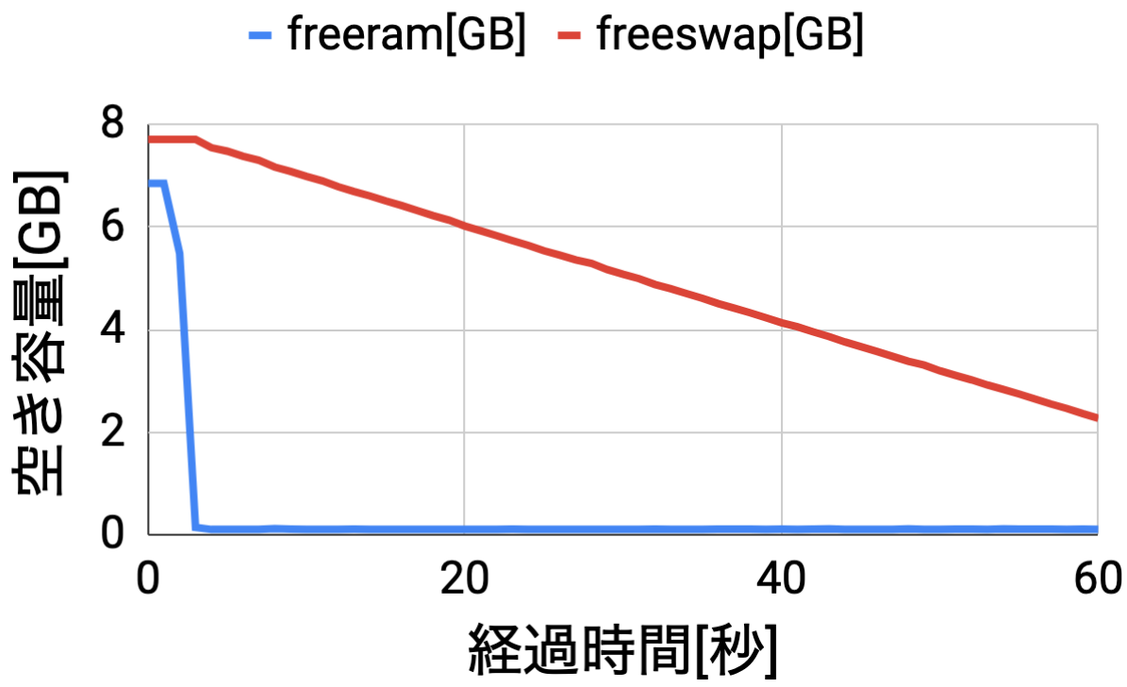
# メモリが枯渇する障害の検知

- **メモリを20GB消費するプログラムを実行**
  - プログラム実行の約1秒後にログ出力が一旦停止
    - ページングが多発しシステムの性能が低下したため

✓それから58秒で障害を検知

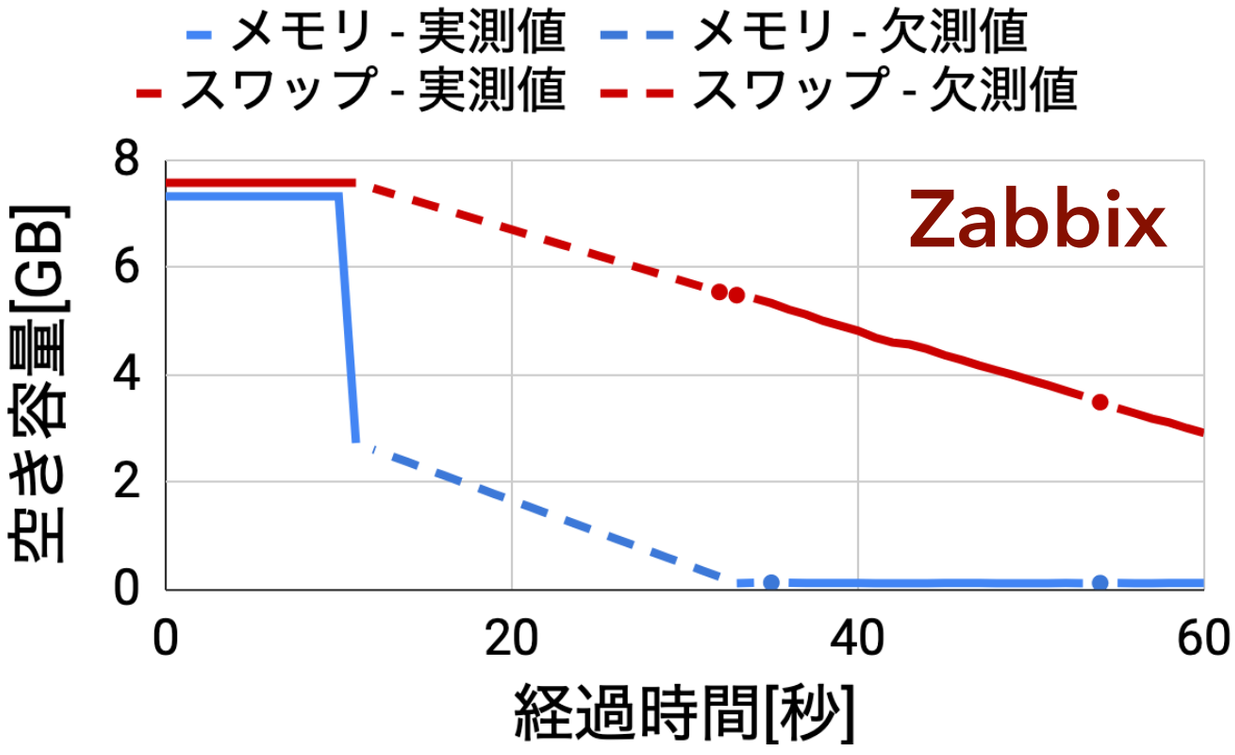
    - スワップを消費するのに時間がかかったため
  - この後、OSによって
    プログラムは強制終了

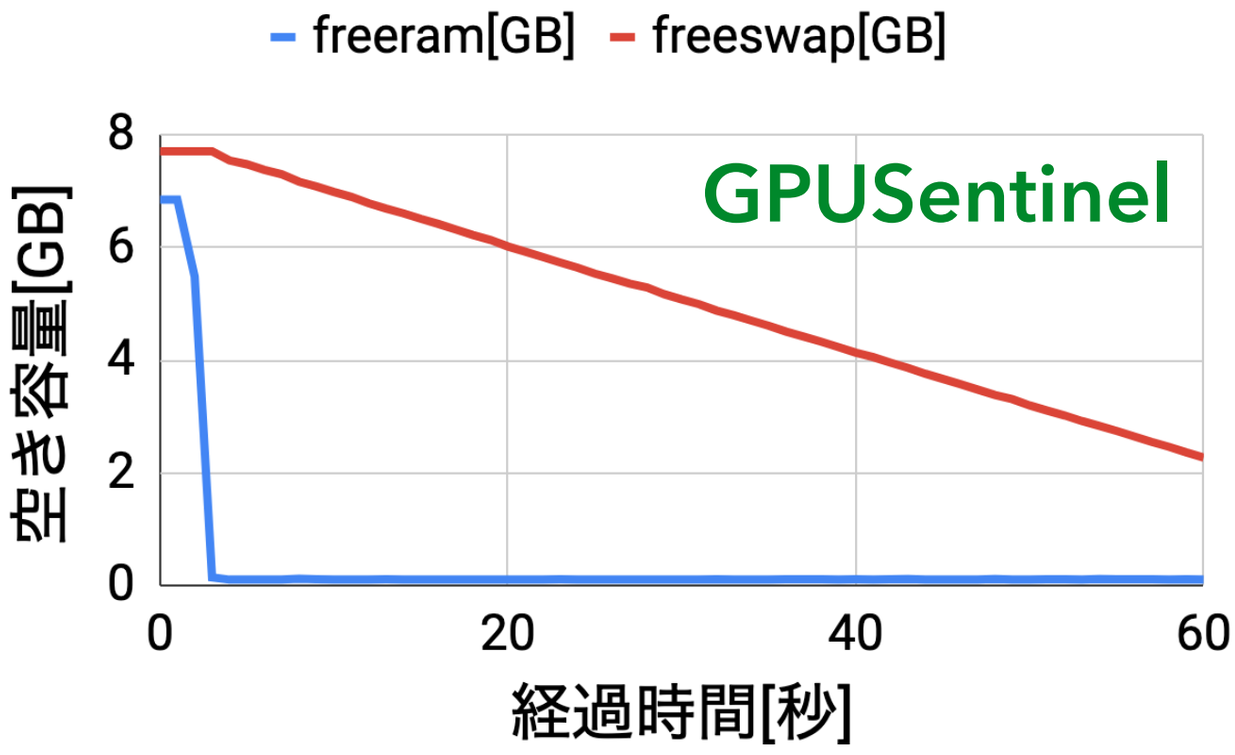
✓OSより早く検知



# Zabbixによるリモート監視との比較

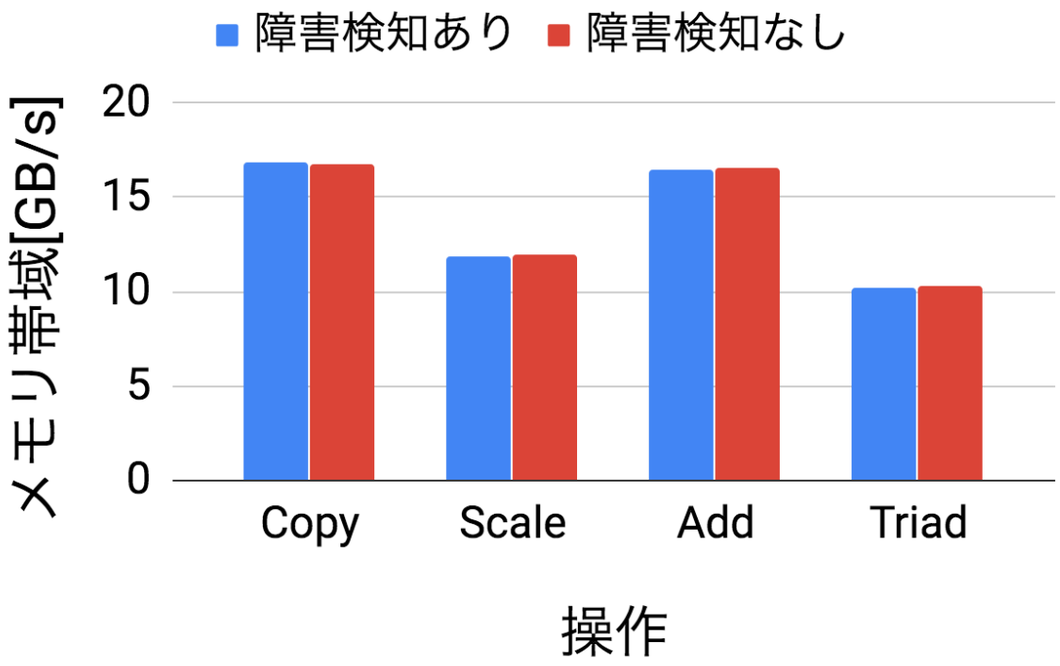
- メモリを枯渇させる実験において、Zabbixを用いてメモリとスワップの空き容量を監視
  - Zabbixでは情報を取得できない期間があった
  - ✓GPUSentinelでは常に情報を取得できた

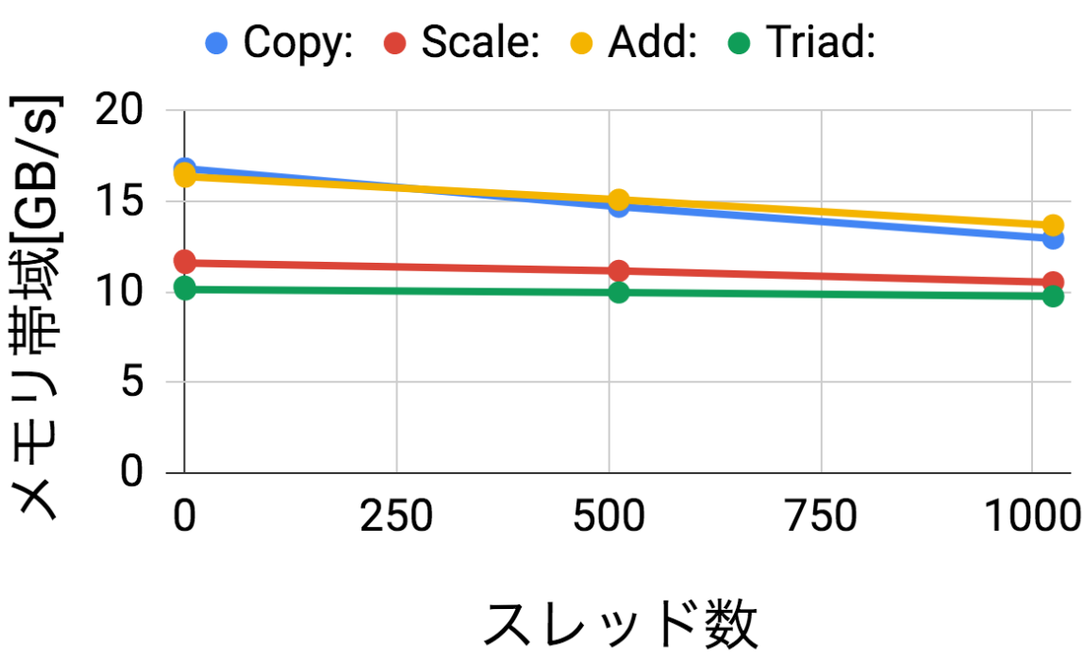


# メインメモリ帯域への影響

- **STREAMに対するオーバヘッドを測定**
  - 本研究で作成した3つの検知プログラムを並列実行
    - ✓ ほぼオーバヘッドなし
  - メインメモリ上のデータのコピーを並列実行
    - ✓ 1024スレッドで20%のオーバヘッド



# 関連研究

- **SHFH [Zhu et al. ’14]**
  - カーネルモジュールで6種類の障害を検知して回復
  - OSが停止する障害は検知できない
- **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
- **SPE Observer [Kourai et al.’12]**
  - Cell/B.E.の隔離されたCPUコアを用いて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**
  - メインメモリ全体をGPUにマッピング
  - LLVMを用いた自動アドレス変換により検知プログラムが透過的にOSデータを監視
  - 障害を検知できることとオーバヘッドについて確認
- **今後の課題**
  - より多くのOSデータを監視することで誤検知を低減
  - OSデータを書き換えることで障害から回復

# 質疑・応答

# OSレベルの障害の事例

- **サイボウズ 2012/6/30**
  - OSにうるう秒を受け取るとシステムが停止する不具合による障害

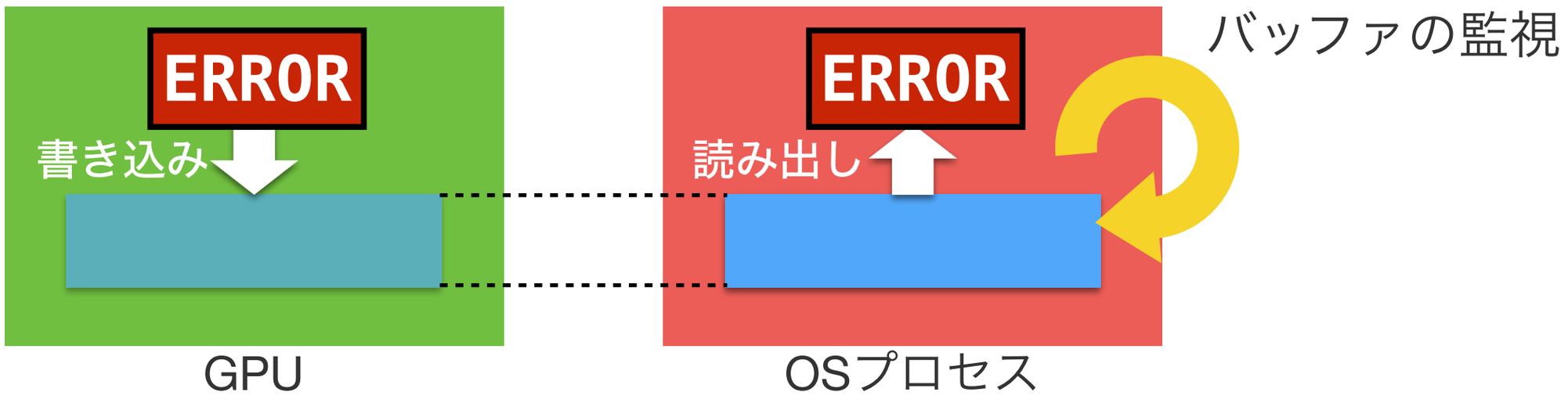
    ×サーバが応答しなくなった
- **Drive Network 2019/1/23**
  - OSファイルシステム不整合による障害

    ×ホームページが参照できない、接続が拒否される
- **Cloud Garage 2018/9/10,11**
  - OSのSwapファイル関連のバグによってプロセスが強制終了される障害

    ×障害が発生したホストOSに収容されている一部インスタンスで通信不能状態

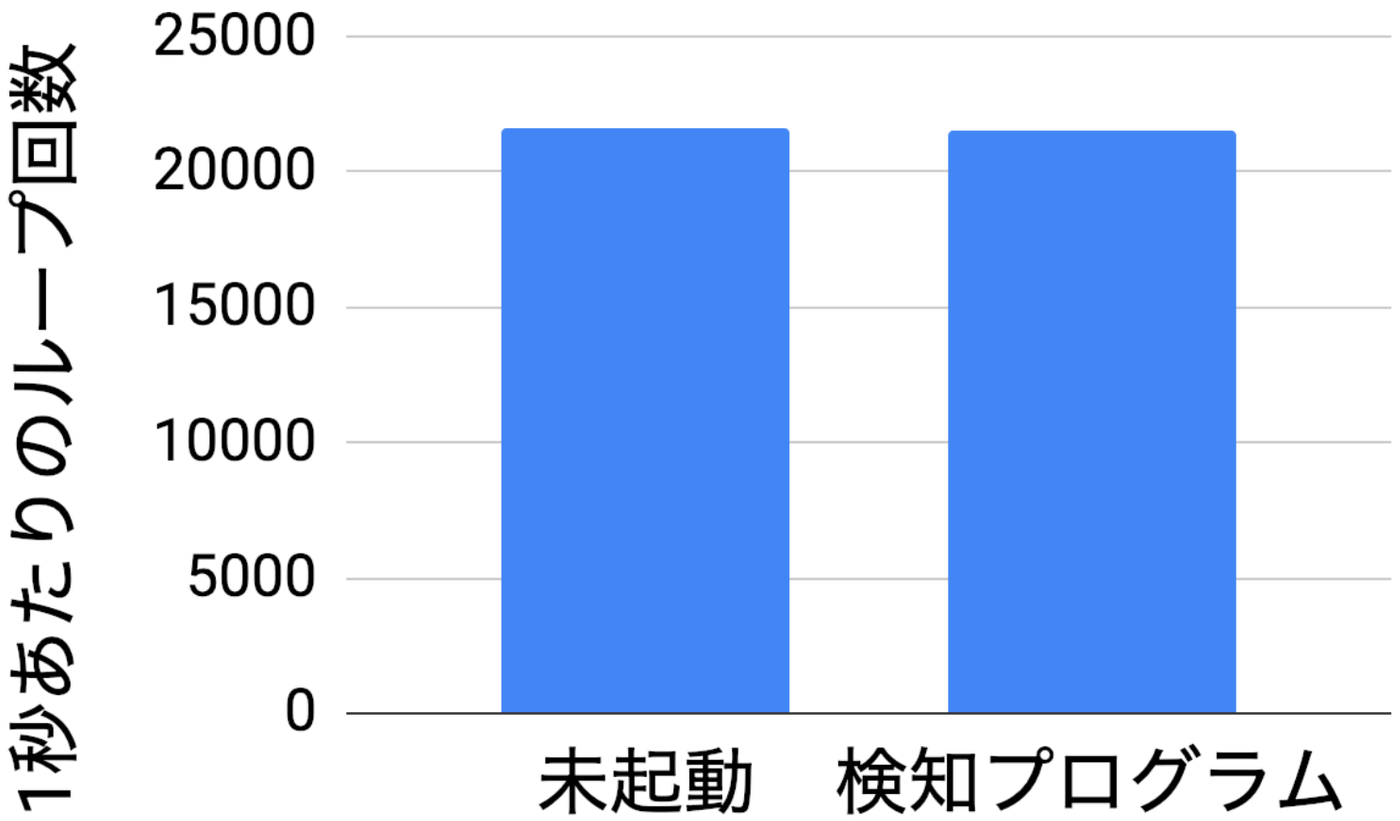
# リアルタイム出力機構の詳細

- ✕ 通常のCUDAではprintfで出力された文字はカーネル関数が終了してから画面に出力される
  - デバッグを行う時に不便
- リングバッファを用いてリアルタイムに文字を出力できる機構を実装



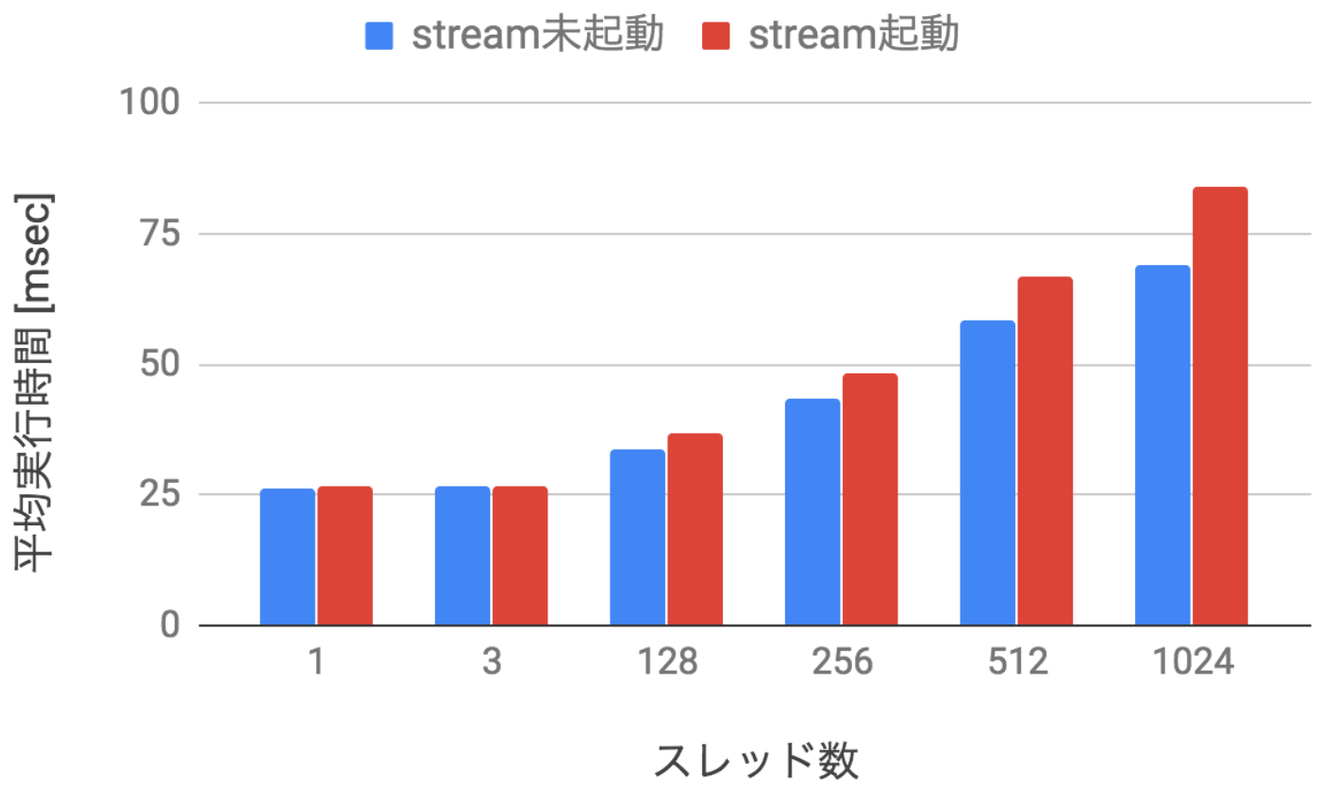
# CPUのオーバヘッド

- **UNIXBenchのDhrystoneを用いてオーバヘッドを測定**
  - 本研究で作成した3つの検知プログラムを並列実行
    - ✓ ほぼオーバヘッドなし



# GPUSentinelへの影響

- **GPUSentinelの処理にかかる時間をSTREAMを動作させながら計測**
  - メモリコピープログラムをスレッド数を変化させながら動作



# 一部CPUデッドロックによるリソース減少

- 2つのCPUをデッドロックさせるカーネルモジュールを挿入
  - ✓5.1秒で検知