Linux のバグと脆弱性
～DEOS からのセキュリティと信頼性へのアプローチ～

河野健二（慶應義塾大学）
光来健一（九州工業大学）
山田浩史（東京農工大学）

Open System におけるセキュリティ

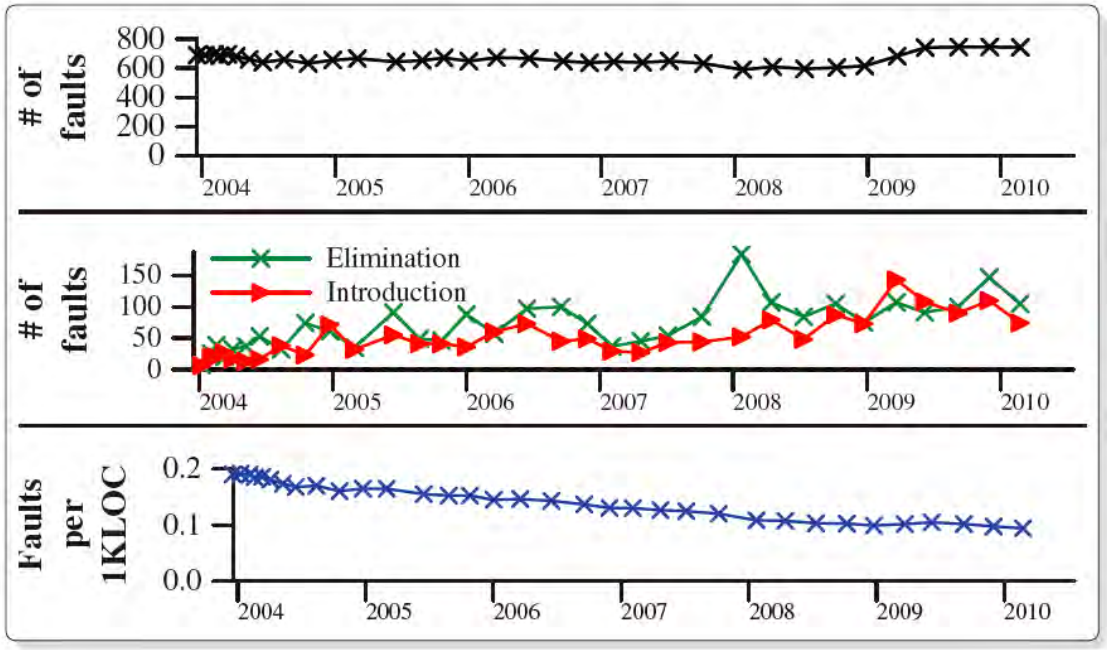
sslab system software

- セキュリティ上の脅威を想定しておくことが難しい
  - システムがオープンであるため・・・
    - これは不特定多数のユーザからアクセスされる
    - ベンダの異なるソフトウェア・コンポーネントが相互に連携する
- 想定外／未知の脅威を“想定”する必要がある
  - DEOS では・・・
    - システムの動作異常を汎用的に捉える必要があり・・・
      - どういう理由で異常が発生しているのかは気にしない・気にできない
    - 異常発生の原因を究明する必要がなく・・・
      - 想定外／未知の脅威の原因究明は時間がかかる
    - システムを正常な状態に復帰できなければいけない

Current Status of Linux

- Linux has 700 bugs (faults) [Palix et al. 2011]
  - Counted are statically detectable, simple bugs
    - e.g., NULL pointer check miss
  - Subtle bugs that lead to race condition or resource leakage are not counted

Faults rate per line is decreasing, but the total # of faults is almost constant



OS Kernel as Buggy Software

sslab system software

- OS のバグを現実として受け入れるべき
  - バグのない OS は事実上，実現不可能
  - カーネルの多機能化・複雑化・環境の多様化
    - 新しい機能はバグが多い
      - ex. RCU ロック
    - 実行環境依存のバグ
      - ex. 特殊なハードウェアの組み合わせで顕在化するバグ
- 環境変化を前提とする DEOS ではなおさら
  - 環境の変化はバグを誘発する
    - 経験的にも，実証的にも

セキュリティの位置づけ

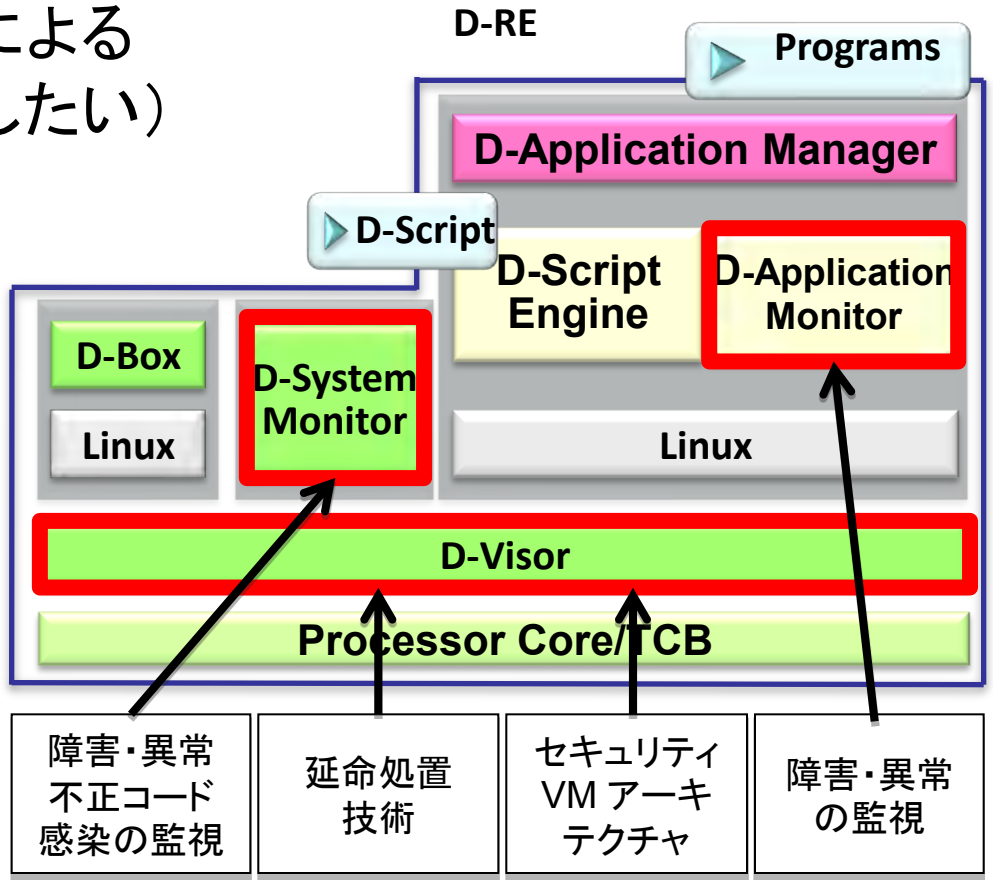
sslab system software

- カーネルバグに起因するセキュリティホールが対象
  - セキュリティとバグが統一的に扱い可能になる
  - 対策としての「ソフトウェアの修正」という点では同じ
    - セキュリティホールの修正はコードの若干の修正である
      - 引数チェックの追加，インデックスチェックの追加
      - 設定ファイルの修正などなど
- 両者の違いは？
  - フェイラ：比較的検出が容易
    - 人間が“おかしい”と認知できるまで問題が表面化する
  - セキュリティ障害：検出が難しい
    - 表面的には“正しく動いている”ように見える
    - 裏でおかしなことが起きている
      - こっそりパスワードが漏洩している

DEOS における考え方

異常検出機構を研ぎ澄ませばよい

- セキュリティ障害も検出可能にする
  - そこから先は D-ADD による
    統一的扱いが可能（にしたい）
- 監視機能の集約
  - D-Visor +
    D-System Monitor
  - D-Visor による
    イベント注入・監視機構
    がキモ



Basic Concept

sslab system software

- Virtually Extending the Life of OS Kernels
  - カーネルに事実上の延命処置を施すことを可能とする
- 延命処置とは？
  - バグを踏んでも OS が動き続けているように見せること
    - Rejuvenation: Software aging を防止する技術
    - Recovery: Failure から回復する技術
    - Dynamic Update: OS の動的なアップデート
    - 機能縮退：健全に動作するサブシステムのみを動かす
    - ….

    などの多くのコンセプトの上位概念
  - エラー状態を修復するとは限らない
    - 再起動したり，カーネル・イメージをすり替えたり・・・
    - もちろん，エラー状態を修復してもよい

Life Extension in a Nutshell

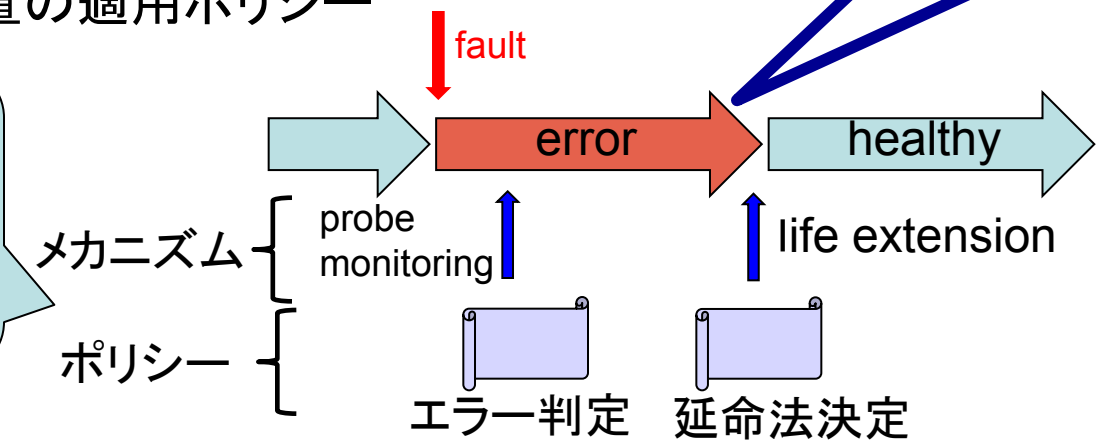
- エラーの種類に応じてさまざまな延命処置が可能
  1. エラー状態の検出
  2. エラー状態に応じて，延命処置手法の選択
  3. 延命処置の適用

- メカニズムとポリシーの分離
  - エラー状態を検知するためのメカニズム
  - 延命処置を行うためのメカニズム
  - エラー状態の判定ポリシー
  - 延命処置の適用ポリシー

エラー状況に応じて何らかの方法で，OSを健全な状態に戻す

・SW のみリブート
・HW からリブート
・機能縮退実行
・etc.

洗練された延命処置ができるよう検知・延命の両技術を開拓



Examples of Life Extension

- 例1: Rejuvenation による software aging の回避
  - Software aging: エラー状態が蓄積してフェイラを起こすこと
    - 例：メモリ・リーク，浮動小数点の丸め誤差の蓄積など
  - メモリリークによる aging の延命
    - メモリ使用量（物理／仮想メモリ）を計測 ← メカニズムとして提供
    - 統計処理により aging の発生を検知 ← ポリシーとして記述
    - システムの再起動を行う ← メカニズムとして提供
- 例2: 振る舞い監視によるマルウェアの検知
  - マルウェアに感染した OS の “振る舞い” は健全な OS とは違う
    - “振る舞い” のずれを検出
  - キーロガーの検出
    - 仮想的なキー入力を与え，I/Oアクセスを計測 ← メカニズムとして提供
    - 統計処理によりキーロガーの存在を検知 ← ポリシーとして記述
    - クリーンなメモリイメージへの差し替え← メカニズムとして提供

Life Extension のための要素技術

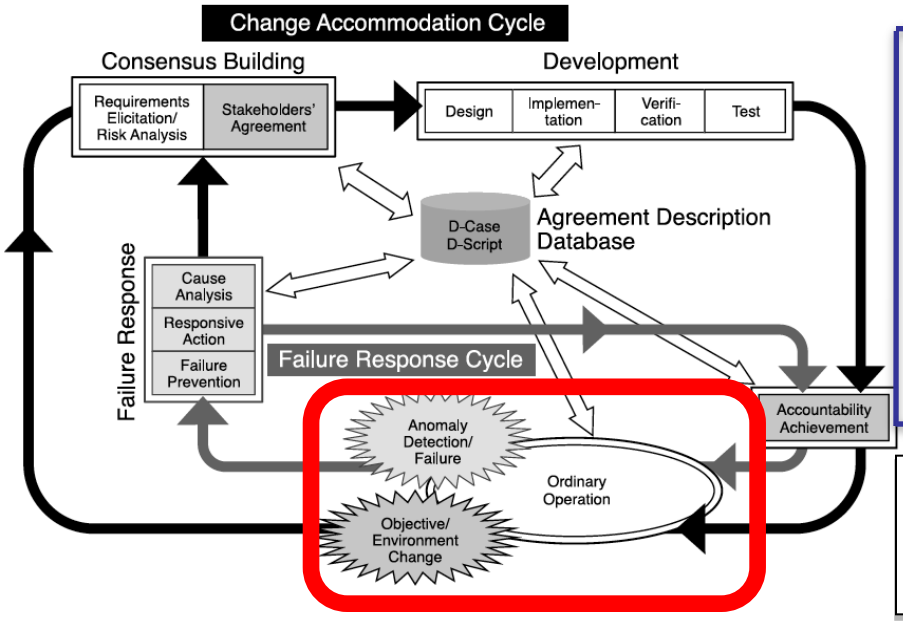
sslab system software

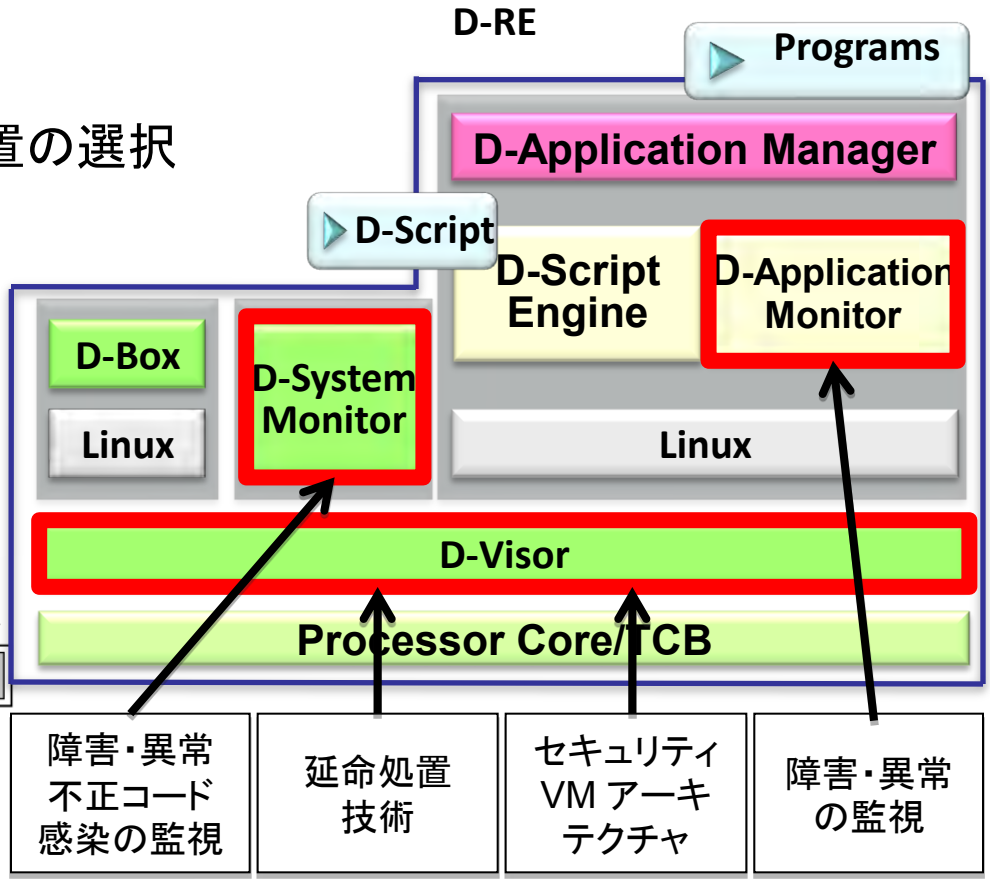
洗練された障害・異常検知と延命処理を可能にする

- Detecting deviant behaviors:
  - “異常” な状態を検出するための技術
    - Kernel crash ← 容易に検知可能
    - Performance anomaly ← 統計的な手法により検知
    - Malware への感染 ← カーネルの振る舞いの矛盾を検知
- Extending life of OS kernels:
  - “異常” な状態を誤魔化すための技術
    - エラー状態に応じたさまざまな高速リブート機能
    - エラー状態の識別によるカーネル縮退実行

DEOS における位置づけ

- 2 重ループを回すための実行時基盤ソフトウェア群
  - D-RE がメカニズムを提供
    - 障害・異常の検知
    - 延命処置のための設定変更，再起動，アップデート等
  - D-Script にポリシーを記述
    - エラー状態の検出方針
    - エラー状態に応じた延命処置の選択

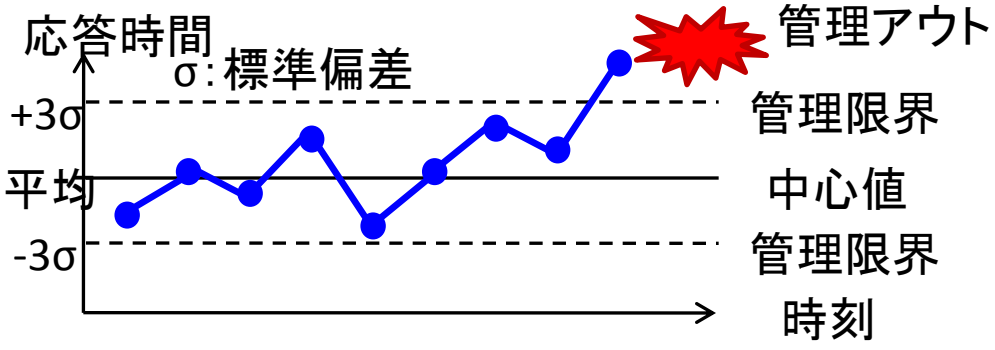


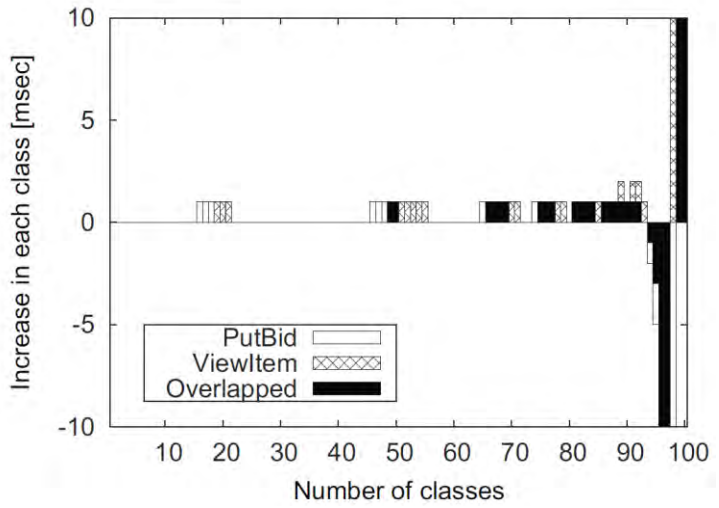


パフォーマンス障害の予兆検知

障害の予兆検知

- 通常運用時のかすかな兆候を見つけ出す技術
  - 異常はなくとも兆候があれば対策がとれる！
- ウェブアプリケーションの応答時間を監視
  - “管理図”という手法の応用により，予兆を検知
    - 管理工学で開発された製造工程の異常を発見する手法
  - “性能異常シグネチャ”により原因究明を支援
    - 異常時の応答時間の変化から，
    - 障害の要因となるコンポーネントの絞り込みを支援





障害の予兆検知：ケーススタディ

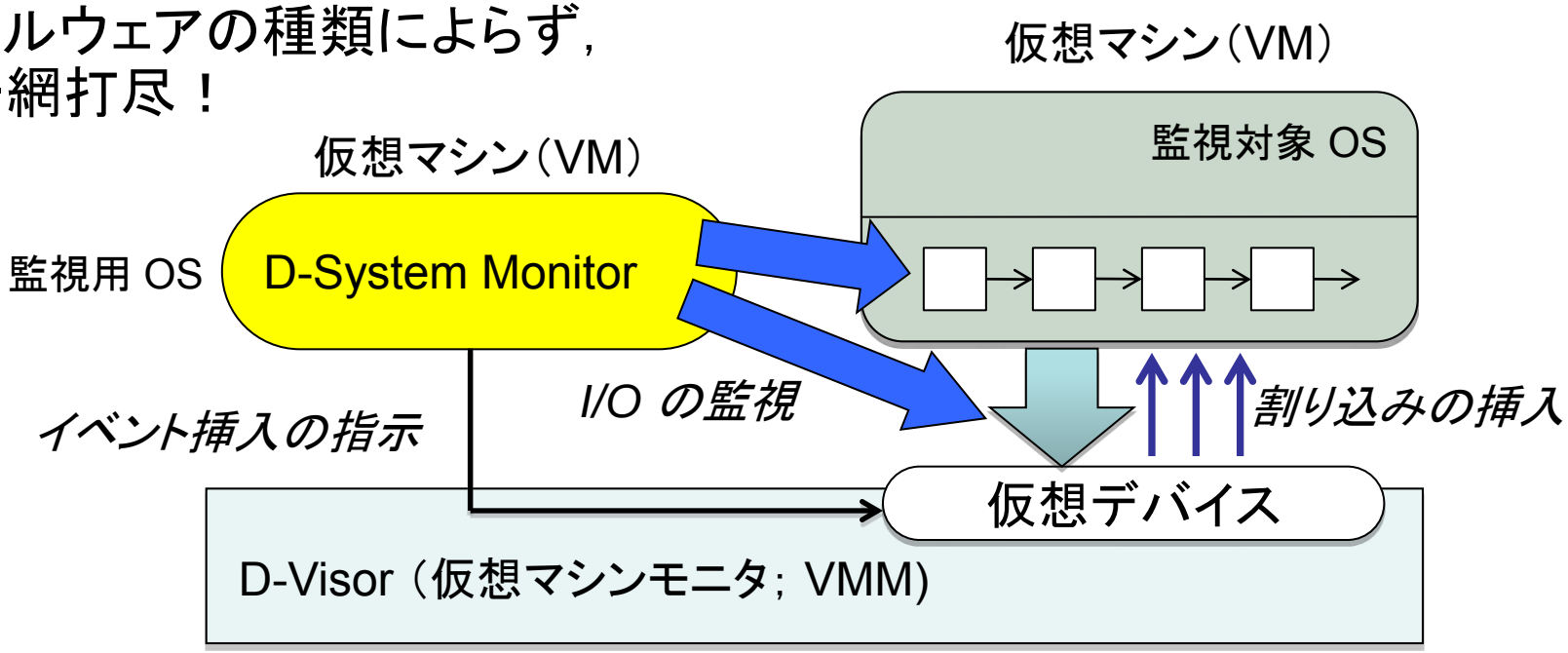
sslab system software

- RUBiS というオークションサイトを利用
  - eBay という商用サイトを模した標準的なベンチマーク
- 管理図により障害の予兆を検知
  - SearchItemsInCategory というリクエストの応答時間が統計的に異常な振る舞いをする
    - 最小応答時間が増加傾向にあることを検知
    - 応答時間を見ていてもとても気づかない微少な変化
- 性能異常シグネチャにより原因究明を開始
  - 障害の発生したコンポーネントを絞り込む
    - SearchItemsByCategory サーブレット
    - SB_SearchItemsByCategory セッションビーン
    - Query セッションビーン
  - 各コンポーネントを調査した結果，Query セッションビーンにバグを発見
    - データベースのインデックスの扱いに不具合があった

カーネルへのマルウェアの検知

カーネルのマルウェア感染検知

- 着眼点：
  - マルウェアに感染した OS の “振る舞い” は何か違うはず
    - 振る舞いにまったく違いがなければ，感染していないのと同じ
- アプローチ：
  - 仮想マシン技術により監視対象 OS の “振る舞い” を監視する
  - 健全な OS の “振る舞い” と何か違えば，
    マルウェアの種類によらず，
    一網打尽！



“振る舞い” 監視による効用

sslab system software

- シグネチャ不要の感染監視機構
  - マルウェアを特徴付けるバイト列に依存しない
- 攻撃ベクタに依存しない
  - マルウェアを仕込む仕組みを監視しているわけではない

したがって，マルウェアが進化しても検出可能！

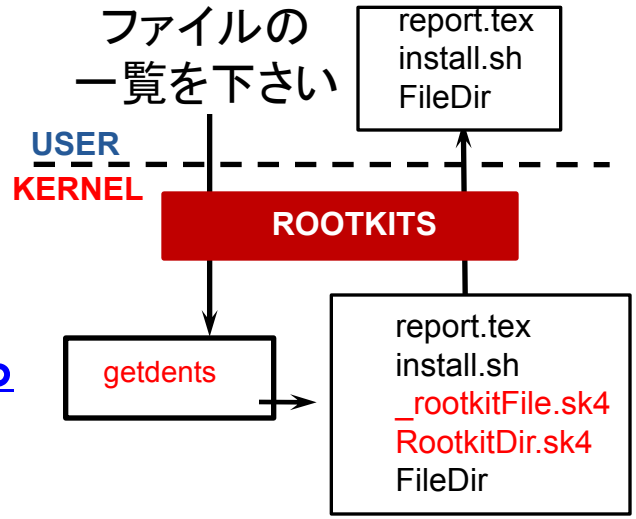
- 新たな “振る舞い” に対してのみ対策をとればよい
  - 亜種対応のために “変化対応サイクル” を回す必要がなくなる

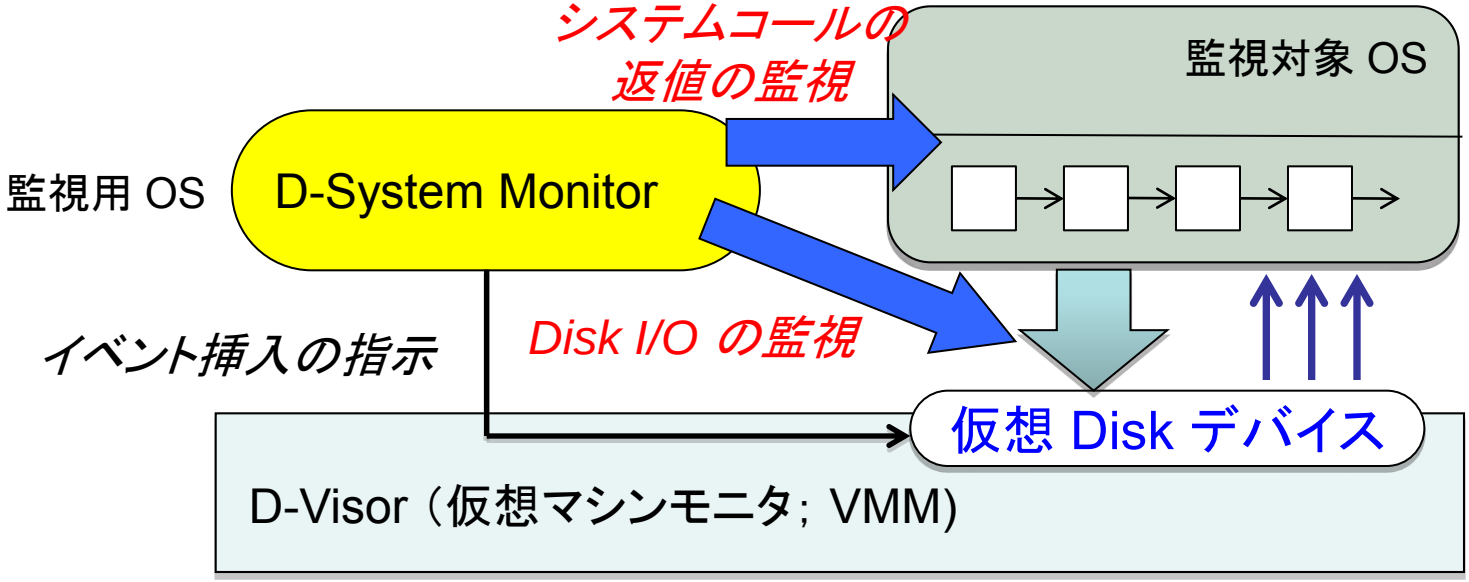
- 以下の “振る舞い” を汎用的に検知できることを実証
  - （典型的な）ルートキット
  - C&C 型のボット
  - キーロガー
  - アドウェア
  - 偽アンチウィルスソフトウェア

“振る舞い”監視の例（その1）

- ファイル隠蔽型ルートキット:
  - 隠蔽したいファイルを一覧から削除する
    - 攻撃方法だけでも多様:
      - Replace/redirect system calls,
      - Patch Kernel Text, Modify Jump tables, ...
- 監視する“振る舞い”
  - システムコールの返値
  - ディスク I/O の内容

一致しなかったら
異常動作





“振る舞い” 監視の例(その1):評価

sslab system software

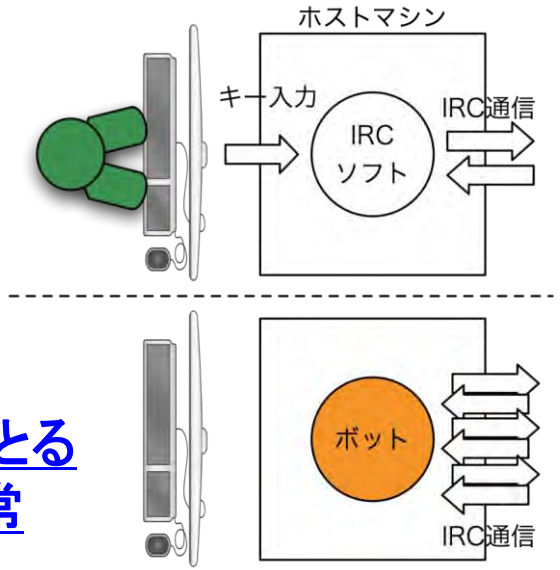
- 実在のルートキット 8 種すべての検知に成功
  - シグネチャ不要
  - 多様な攻撃手法に対して有効

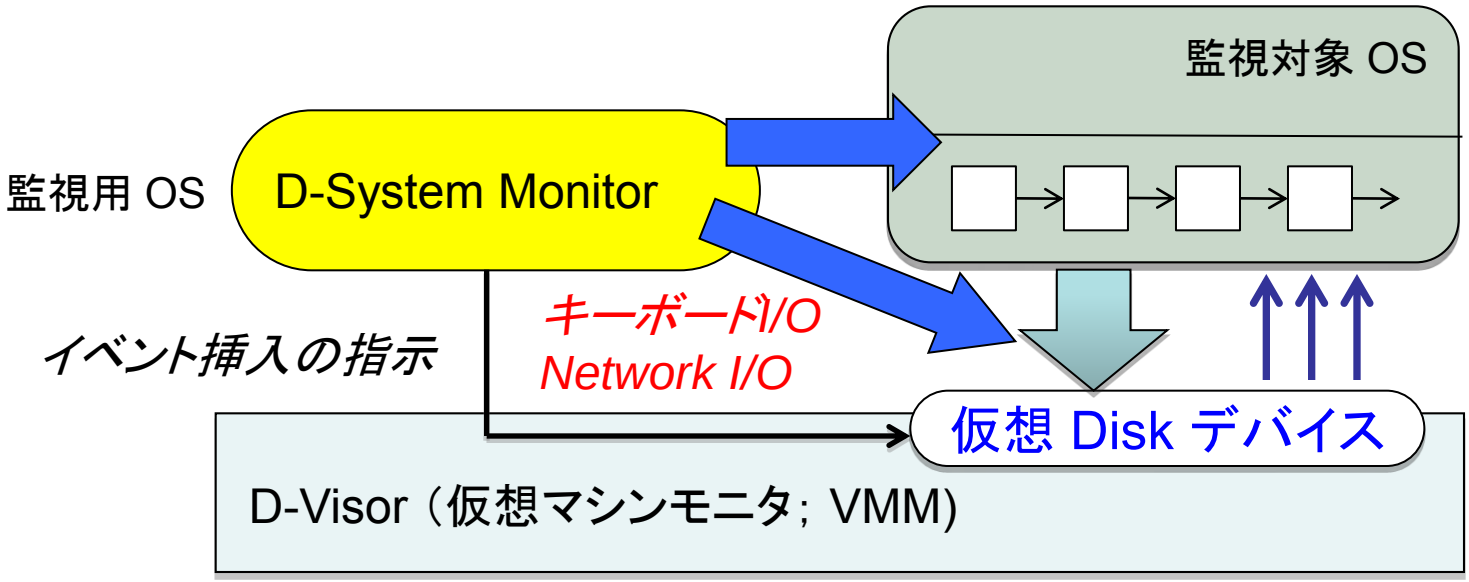
Rootkit	Attack Vector	Target	Detected
adore – 0.42	LKM	Sys. Call Table	Yes
rial	LKM	Sys. Call Table	Yes
enyelkm	LKM	Kernel Text	Yes
override	LKM	Sys. Call Table	Yes
adore-ng0.56	LKM	Virt. File System	Yes
mood-nt	Raw mem. access	Sys. Call Table	Yes
superkit	Raw mem. access	Kernel Text	Yes
suckit2priv	Raw mem. access	Kernel Text	Yes

“振る舞い”監視の例（その2）

- ボットへの感染を検知
  - ボットの多くはチャット用のプロトコル (IRC) を使って動作する
  - ボットのやりとりする“チャット”は，人間より応答が速いはず
- 監視する“振る舞い”
  - キー入力
  - IRC 通信の応答性

} キー入力と通信の相関をとる
人間と違う相関性なら異常

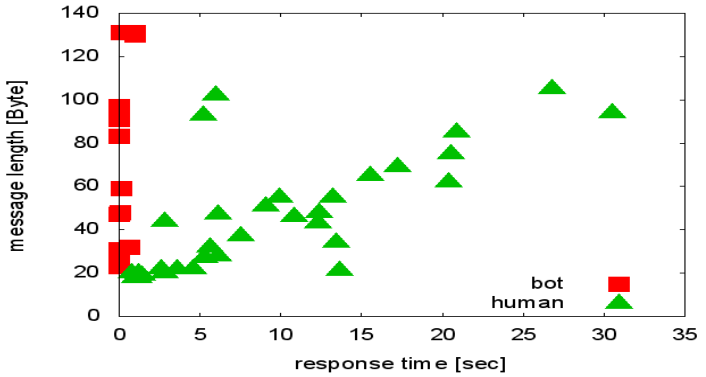




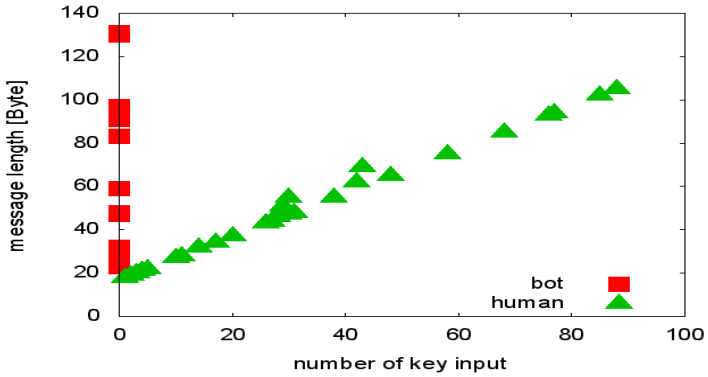
“振る舞い” 監視の例（その2）: 評価

- 22 個の実在のボットに対して検知実験を行った
  - 一部の検体は通信に遅延を入れていた
  - 応答時間だけではなく，キー入力も監視しているため検知に成功

Bot	Detected	遅延挿入
Agobot の系列 3 種	Yes	No
SDbot の系列 4 種	Yes	Yes/No
Rxbot の系列 4 種	Yes	Yes/No
Urxbot の系列 3 種	Yes	Yes
Spybot の系列 4 種	Yes	No
Sbot-RASpreader	Yes	No
shadowbot	Yes	No
darkbot v6a3	Yes	Yes
RAGEBOT	Yes	No



ボットと人間での応答時間のふるまい

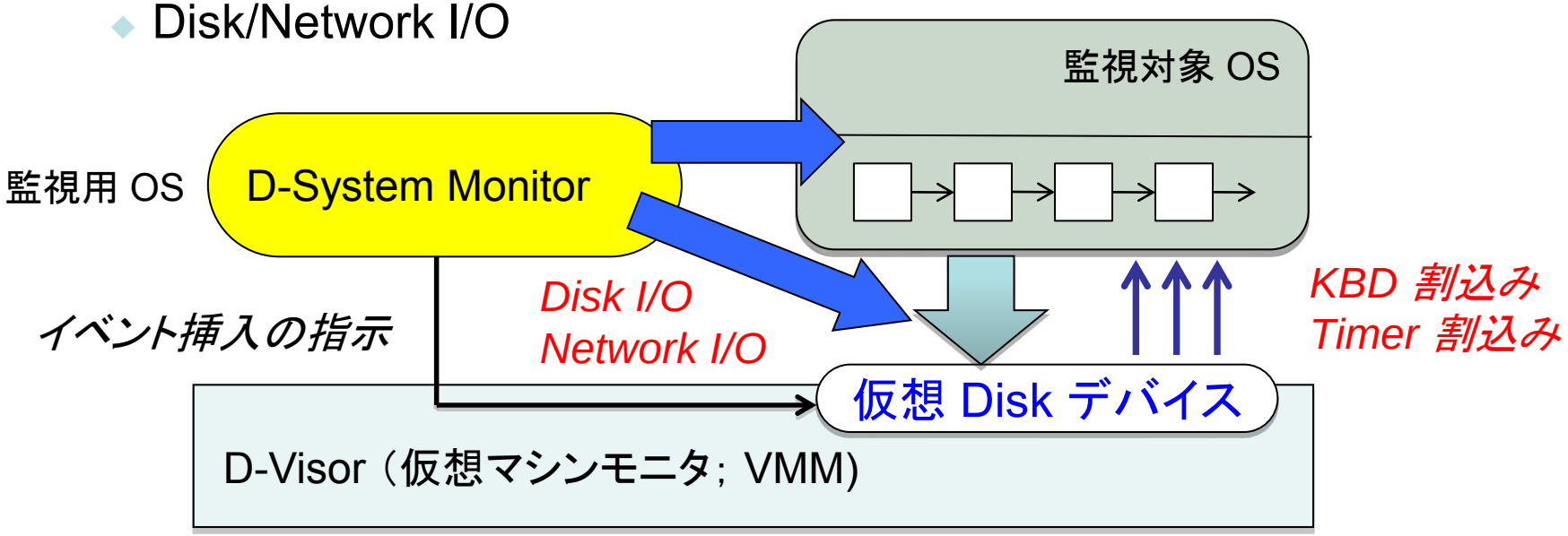


ボットと人間のでキー入力数のふるまい

“振る舞い” 監視の例（その3）

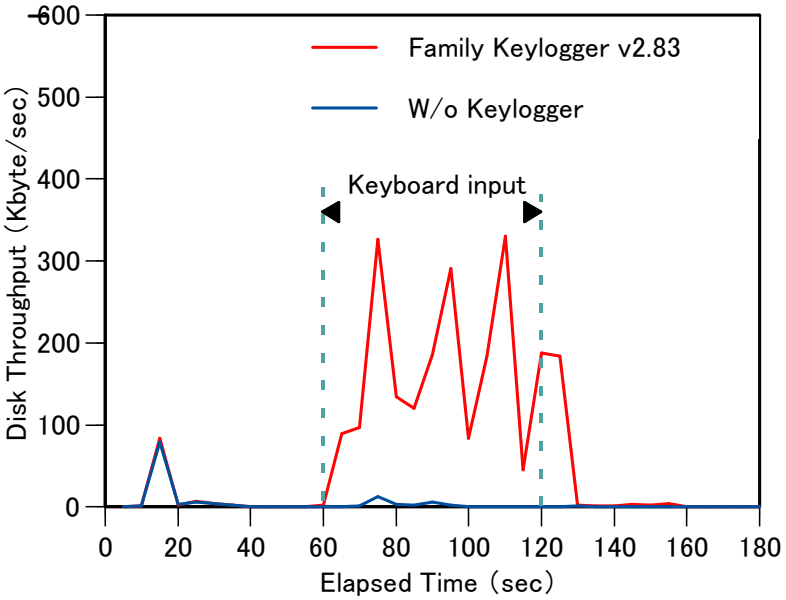
- キーロガーへの感染を検知
  - キー入力を盗む悪質なスパイウェア
  - 処理が軽いため，Passive な監視では検出不能
- 対策：
  - キーロガーを活性化するため大量のキー入力をねつ造
  - 監視する “振る舞い”
    - Disk/Network I/O

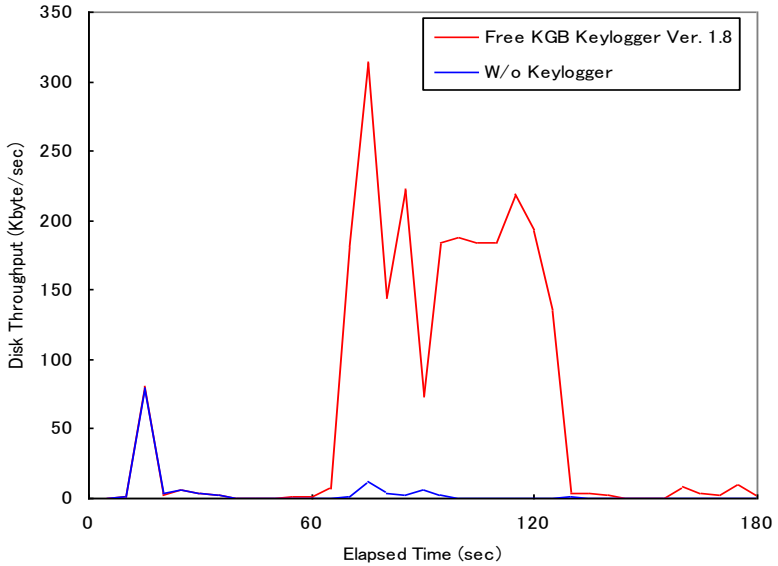
キー入力と I/O の相関をとる



“振る舞い” 監視の例（その3）: 評価

- 実在の 56 種類のキーロガーの内，55 種類を検知
  - 検知できなかったキーロガーは検査時間を長くすれば検知できた
- False positive 無し
  - キーボードユーティリティはほとんど出力しないため誤検知されなかった
- ネットワーク出力は計測されなかった
  - 取得した情報を直ちにネットワークに送信する仕様ではないため





“振る舞い”監視（その他）

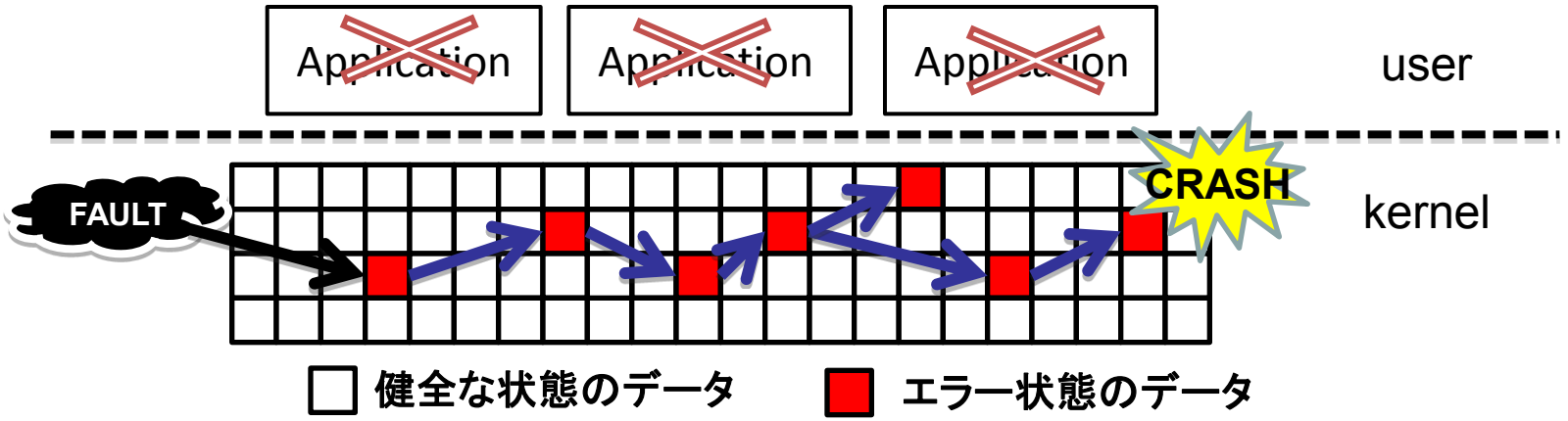
sslab system software

- アドウェアの検知・解析にも利用可能
  - アドウェアとは，ターゲット広告のための情報を盗むマルウェア
  - アドウェアの検知だけではなく，どのような情報を漏洩させているかも解析可能
    - 漏洩されている情報の多くは暗号化されているため，通信内容を監視するだけではわからない
- 偽アンチウィルスソフトウェアの検知にも利用可能
  - 偽アンチウィルスソフトウェアとは，
    - 利用者にウィルスが検知されたとウソを言い，
    - 駆除のためのソフトウェアを売りつけるのに使われる
- マルウェア解析ツール "Yataglass" の開発
  - マルウェアの多くはデバッガでは解析できない
    - 暗号化，難読化が行われている
    - デバッガにアタッチされていることを検知し，振る舞いを変える
  - 記号的実行 (symbolic execution) による強力な解析

カーネルの縮退動作に向けて

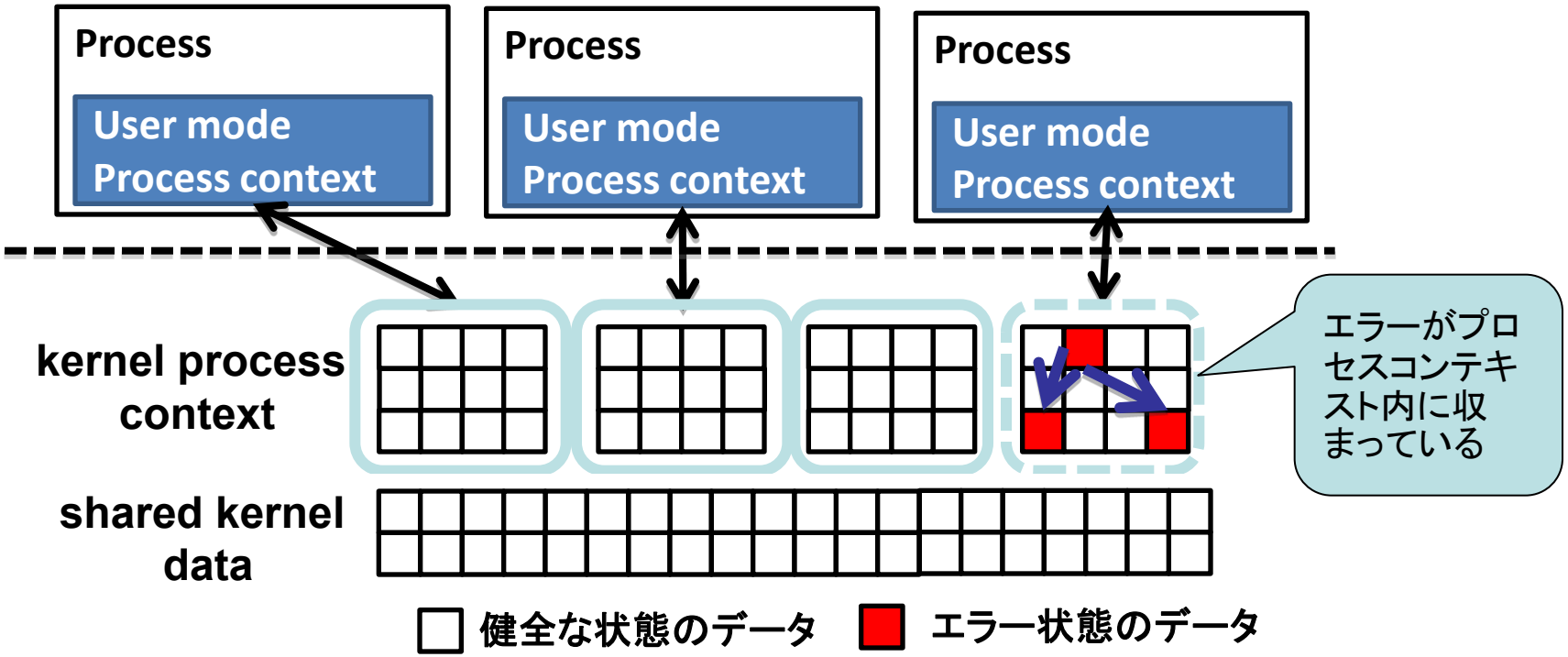
カーネルクラッシュ

- カーネルクラッシュはエラーが伝播して発生する
- カーネルクラッシュの回避・回復は困難である
  - 伝播したエラーから完全回復することは難しい
    - エラーをアイソレートするためにカーネルの改変が必要
    - 大きなオーバヘッドを伴うことが多い [Lenharth et al. '09]
- 問題提起
  - ホントにカーネル全体が壊れてしまうの？
  - 一部しか壊れていなければなんとかなるのでは？



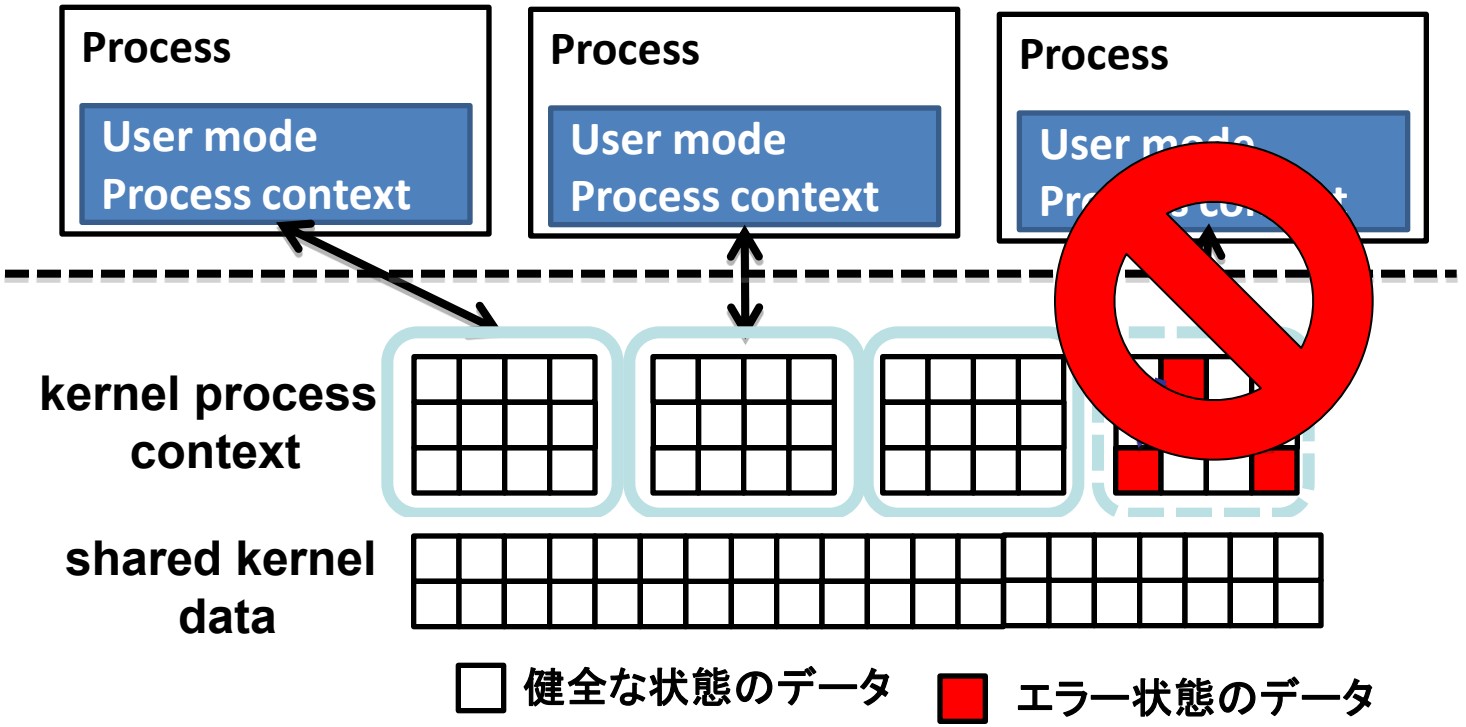
これまでの知見から・・・

- エラー状態は一部に閉じ込められることが多そう
  - エラー伝搬は fault を踏んだプロセス・コンテキストに閉じる
  - となれば，他のプロセスは正常に実行可能
- なぜなら，カーネル・コードは defensive に記述されている
  - 共有データは一旦ローカルにコピーしてから操作．変更後にコミット
  - きめ細やかなエラーチェック



エラー伝播が限定されていれば？

- カーネルを縮退して動作させることが可能
  - エラーの起きた部分だけを切り離せばよい
- 例：エラーがプロセスコンテキストに閉じていたら？
  - フォールトを踏んだプロセス・コンテキストを捨てればよい
  - 他のプロセスは問題なく動作を継続できる



カーネルの縮退動作

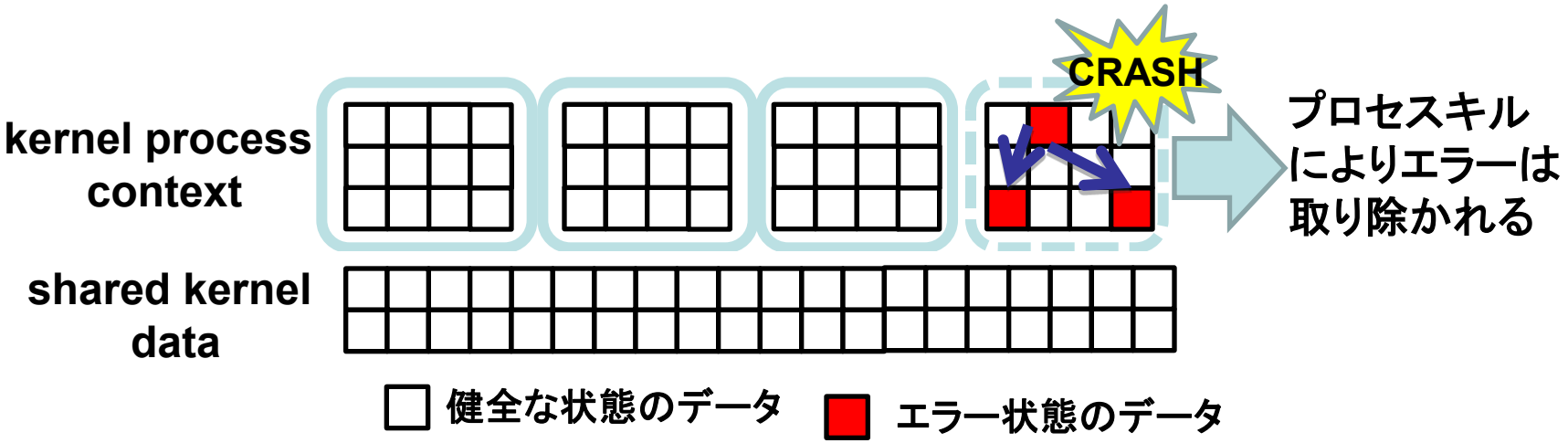
sslab system software

- Life Extension を実現するための主要技術のひとつ
  - エラー状態のコンテキストを切り離し，
    機能の限定された状態で延命する技術

- カーネル内のエラー伝播を詳細に調べる必要あり
  - Linux 2.6.38 にフォールトインジェクション
  - エラー伝播のスコープを調査する
    - プロセスローカル：プロセスコンテキストにエラーが閉じる
    - カーネルグローバル：共有データにエラーが伝播する
  - エラーの多くがプロセスローカルのとき，
    - プロセスキルのみでシステムの稼働を継続できることが期待できる

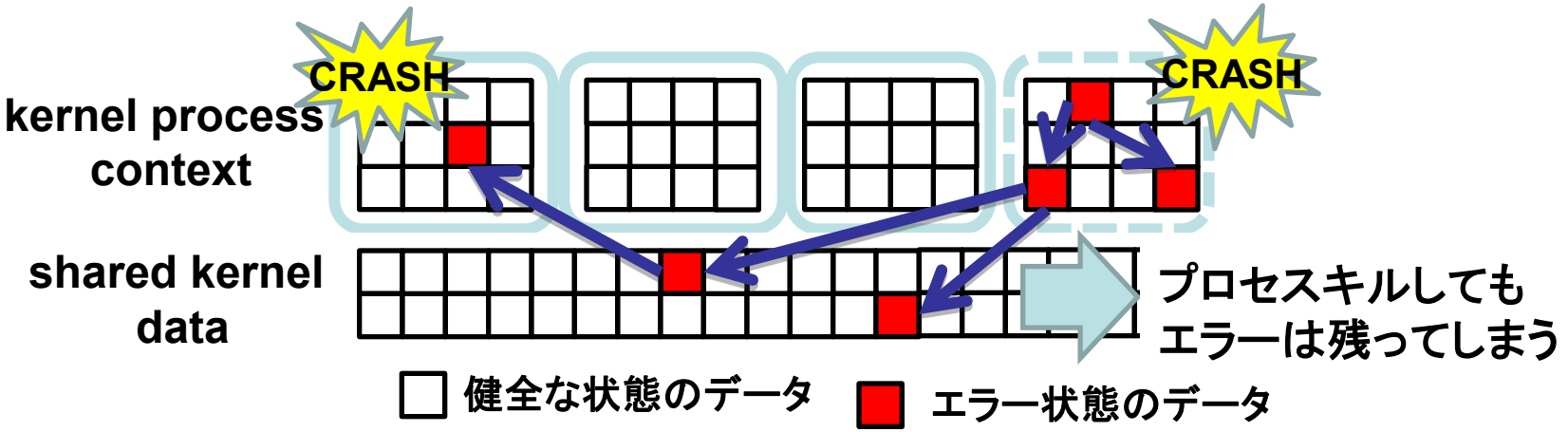
プロセスローカルエラー

- 単一プロセスのコンテキスト内でのみ伝播するエラー
  - プロセスごとにあるカーネルスタックやレジスタにあるデータの破壊を指す
- 他のコンテキストの稼働を継続することができる
  - プロセスキルによりエラーを取り除くことができる



カーネルグローバルエラー

- カーネル内で共有するデータに伝播するエラー
  - プロセス，メモリ等の各種ディスクリプタを表す構造体メンバの破壊など
- 全てのコンテキストがエラーとなる恐れがある
  - プロセスキルしてもエラーは取り除けない
  - システムを停止するべき状況となることが多い
    - ファイルシステム破壊など，何が起こるか予測できない場合



人為的クラッシュによる調査結果

- 104 件のクラッシュが発生した
  - プロセスローカルエラー：70 件
  - カーネルグローバルエラー：34 件
    - オーバラン，リスト破壊，関数ポインタの破壊等による
- クラッシュが起きても，約 7 割の確率でシステムの稼働を継続できることが期待できる

	bcopy	branch	dstsrc	init	inverse	loop	null	off by one	ptr	size	var	total
process-local	5	0	12	10	2	8	10	3	17	1	2	70
kernel-global	13	1	10	1	3	0	0	0	4	1	1	34
total	18	1	22	11	5	8	10	3	21	2	3	104

まとめ

sslab system software

- Life Extension のための技術を洗練していく
  - カーネルの縮退実行
    - Gilles & Julia @ LIP6 との共同研究に発展
    - Roberto @ Napoli も加わる予定
    - 現在，現実の Linux バグのフィールド調査中
      - それを踏まえて具体的な方法を策定
  - 高速リブート機能
    - Kishor @ Duke との共同研究に発展
    - 高速リブート機能の詳細評価、スマートフォン向け高速リブート機能
  - 原因究明のための支援技術
    - 実アプリケーションを模したベンチマークを対象に適用してみる
- D-Visor を支える技術も継続して研究･開発する
  - 監視対象 VM のメモリ暗号化
  - D-System Monitor 用 VM の提供
  - など
- D-ADD との API を策定
  - 障害予兆検知機構を題材に検討していく
    - 永山チームの Web アプリケーションを対象に組み込む
    - フォールトインジェクトしながら API を固めていく