サーバソフトウェアのための動的なセーフティネットの研究

博士3年　光來健一
指導教官　清水謙多郎

ソフトウェアの安全性

- ソフトウェアは複雑化、肥大化している
  - 例：ウェブシステム
    - 初期は静的コンテンツのみ
    - 現在はCGIやJAVAなどの動的コンテンツが増大
  - バグの増大
- ソフトウェアの安全面での問題が顕在化してきている
  - ソフトウェアの持つ権限を悪用する**攻撃**
  - ソフトウェアに残る**不安定さ**

攻撃の脅威

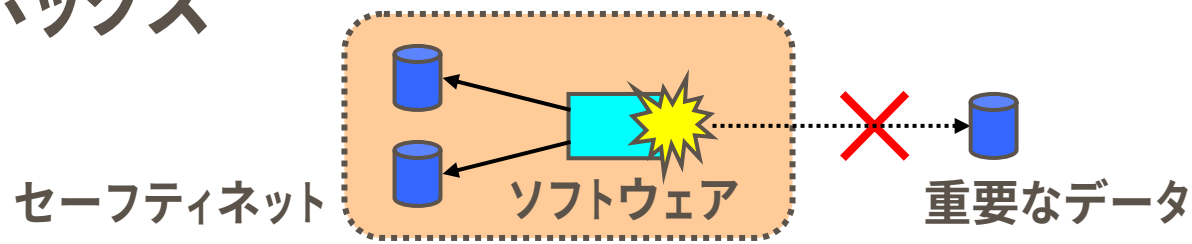
- インターネットに接続するソフトウェアは常に攻撃の危険にさらされている
  - バッファオーバフロー攻撃によるサーバ乗っ取り
  - 悪意あるモバイルコードの実行
- 攻撃を防ぐ様々な手法が提案されているが、未知の攻撃を防ぐことはできない
  - StackGuard [Cowan et al.'98]
    - スタックオーバフロー攻撃を検出する
    - 提案された後に防げない攻撃が見つかった

不安定なソフトウェア

- ソフトウェアの中には不安定なものも存在する
  - β版のソフトウェア
    - 正式版でさえ不安定なものもある
  - フリーソフト
- 言語やコンパイラによるサポートもあるが、必ずしも十分とはいえない
  - PCC [Necula et al.'96]
    - プログラムの安全性を静的に検証する
    - すべてのプログラムを検証できるわけではない

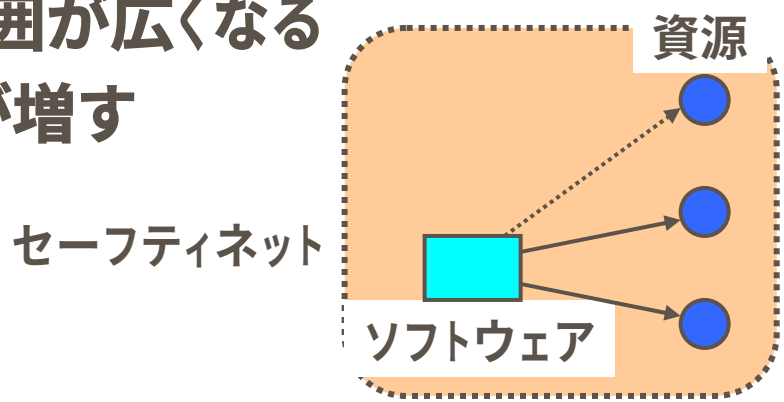
セーフティネットによる防御

- セーフティネットとは？
  - ソフトウェアが万一おかしな動作をした場合でも、被害を最小に抑える防衛ライン
    - セーフティネットの適用範囲はできるだけ狭くすることが重要
- セーフティネットの例
  - サンドボックス



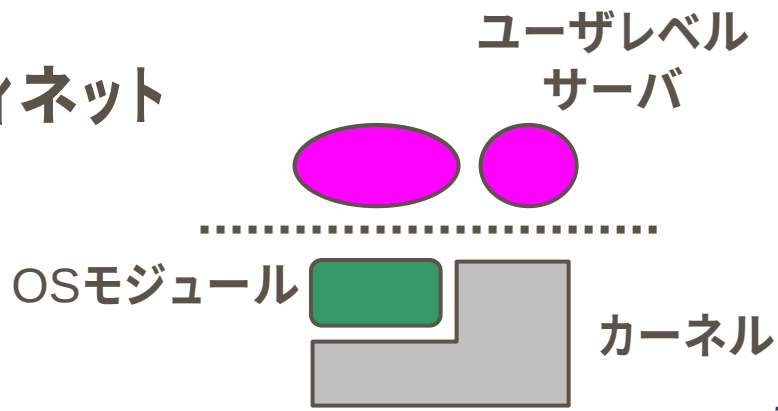
従来のセーフティネットの問題

- 状況によって必要な資源が変わるソフトウェアには適さない
  - 安全性や性能のためセーフティネットは固定
  - 必要となり得る全ての資源をセーフティネットが含まなければならない
    - セーフティネットの範囲が広くなる
    - 万一の場合の危険が増す



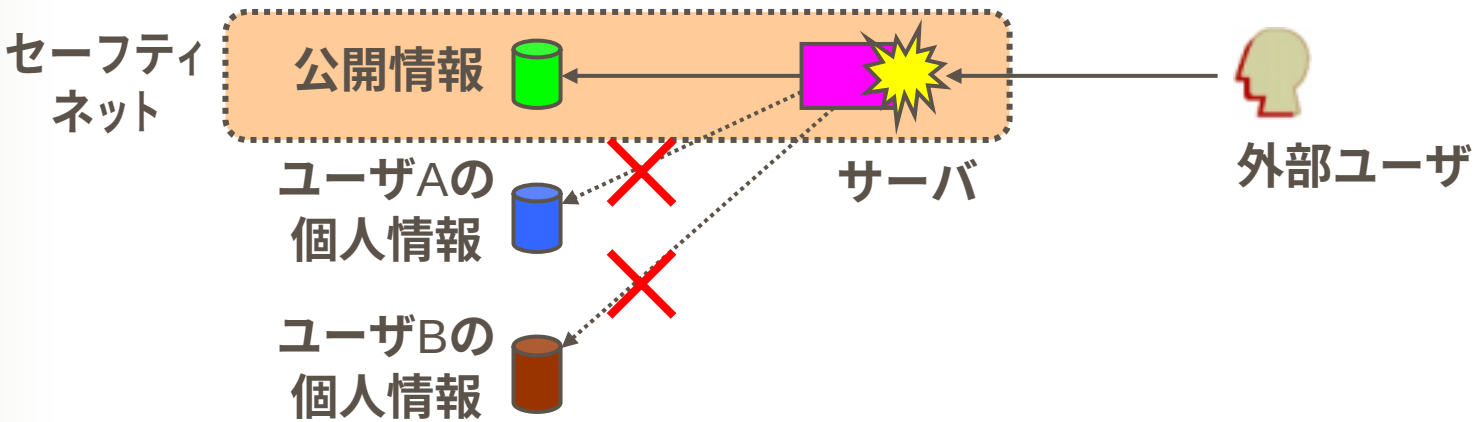
本研究の目標

- サーバソフトウェアに焦点
  - 様々なクライアントに同時に使われる
  - 性能と安全性のバランスが重要
- 動的なセーフティネットの構築
  - クライアントに応じたセーフティネット
    - ユーザレベルサーバ
  - 安定度に応じたセーフティネット
    - OSモジュール



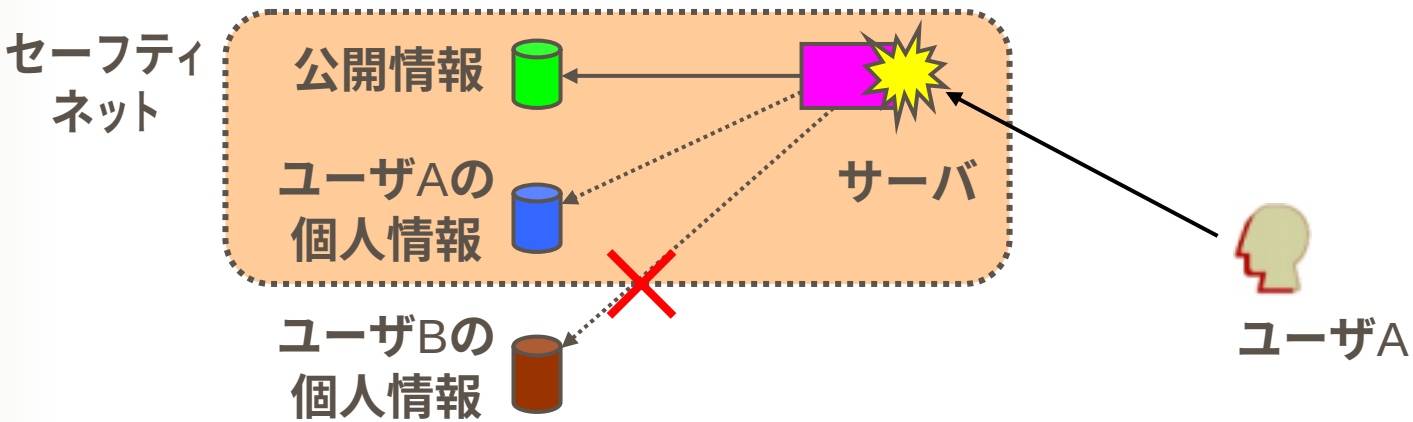
クライアントに応じたセーフティネット

- アクセスしてくるクライアントに応じてサーバのアクセス権限を変える
  - サーバが常に全てのユーザの個人情報にアクセスする権限を持つ必要はなくなる



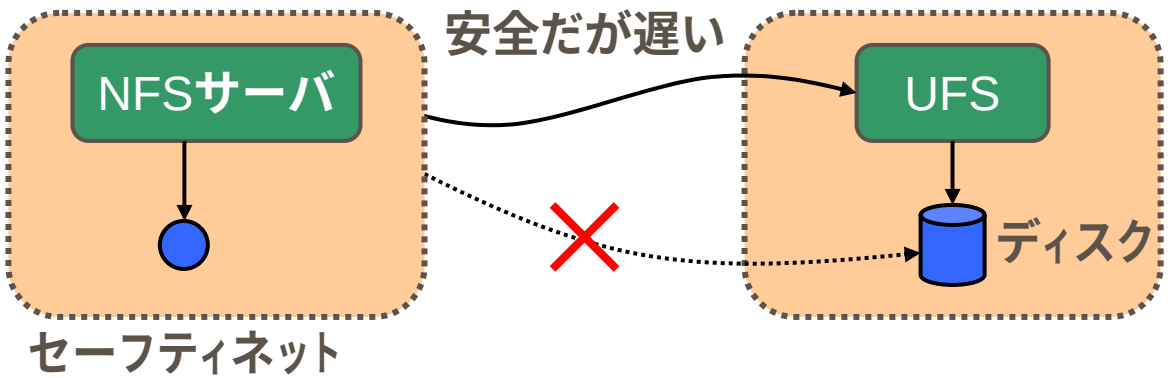
クライアントに応じたセーフティネット

- アクセスしてくるクライアントに応じてサーバのアクセス権限を変える
  - サーバが常に全てのユーザの個人情報にアクセスする権限を持つ必要はなくなる



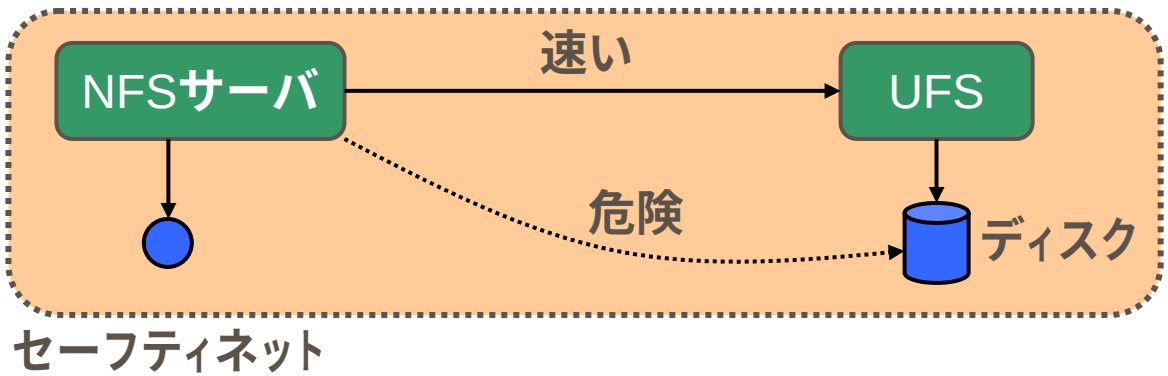
安定度に応じたセーフティネット

- OSモジュールの安定度に応じて保護の仕方を変える
  - 最も不安定なOSモジュールに合わせて強力な保護をする必要がなくなる



安定度に応じたセーフティネット

- OSモジュールの安定度に応じて保護の仕方を変える
  - 最も不安定なOSモジュールに合わせて強力な保護をする必要がなくなる



本研究の主な成果

（I）ユーザレベルサーバ用のアクセス制御機構

- クライアントに応じたセーフティネットを実現
- **プロセスクリーニング**[Kourai et al.'01]によりアクセス制限の安全な変更を実現

（II）OSモジュール用の保護機構

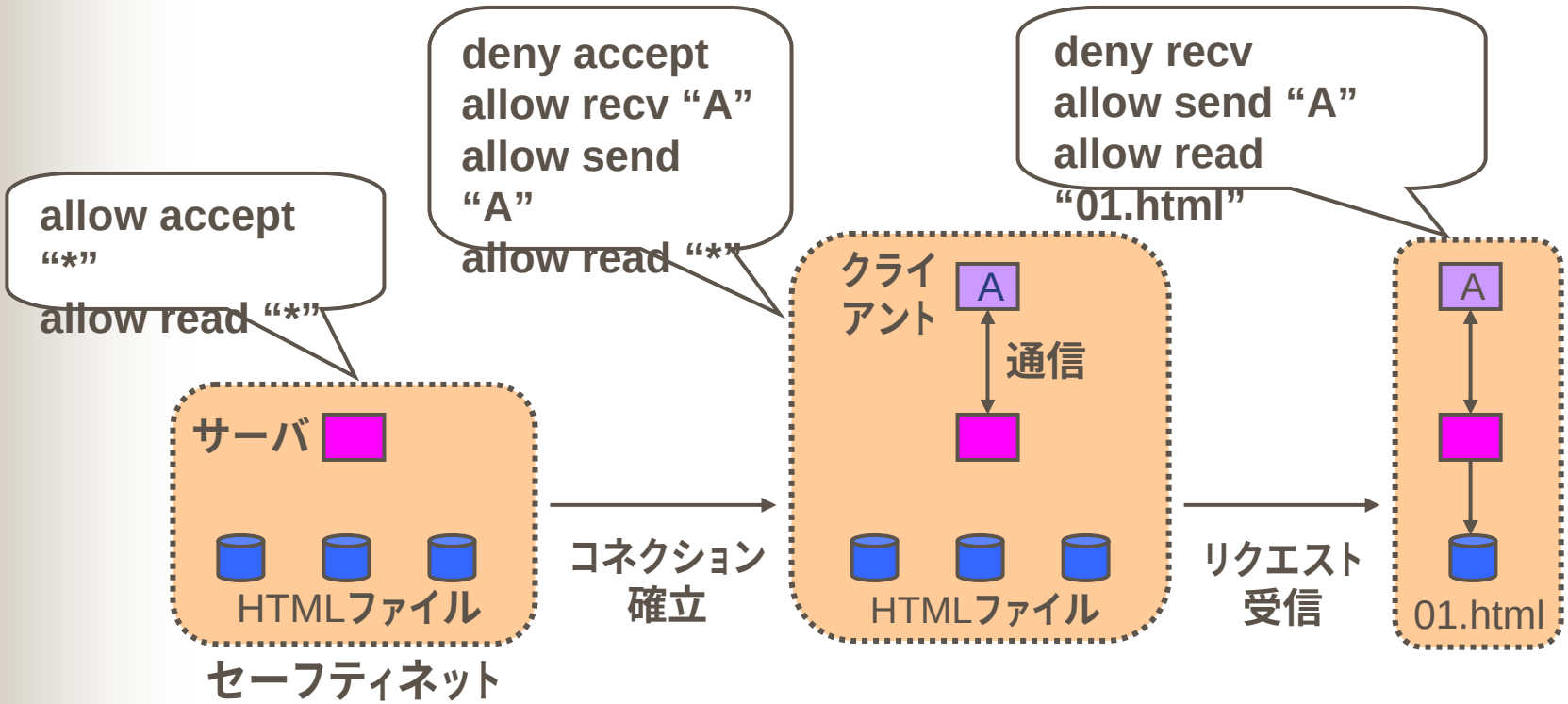
- 安定度に応じたセーフティネットを実現
- **マルチレベル・プロテクション**[Kourai et al.'98]により性能と安全性のトレードオフを実現

（１）ユーザレベルサーバ用のアクセス制御機構

～プロセスクリーニング～

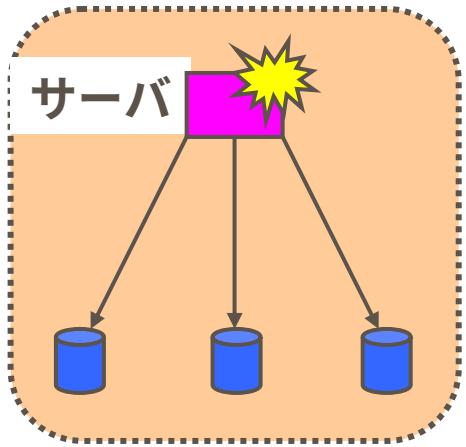
アクセス制限による セーフティネットの構築

- ポリシールールに従ってシステムコールレベルでアクセスを制限する



アクセス制限の変更に伴うリスク

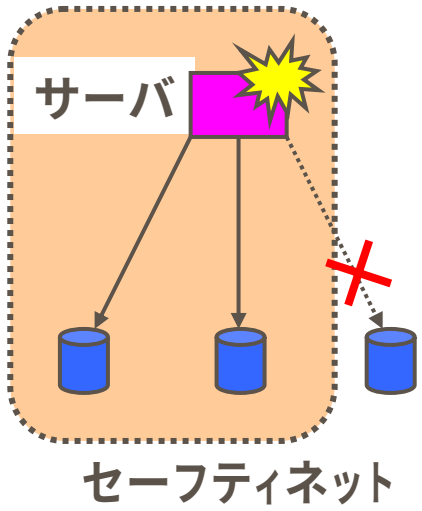
- 攻撃を受けたサーバにアクセス制限の変更を許すのは危険
  - **乗っ取られたサーバ**のアクセス制限が解除されると、全権限を奪われる
  - アクセス制限の変更後に**トロイの木馬**が実行されると、サーバの権限を乱用される
- しかし、攻撃を受けたかどうかを判断するのは難しい



セーフティネット

アクセス制限の変更に伴うリスク

- 攻撃を受けたサーバにアクセス制限の変更を許すのは危険
  - **乗っ取られたサーバ**のアクセス制限が解除されると、全権限を奪われる
  - アクセス制限の変更後に**トロイの木馬**が実行されると、サーバの権限を乱用される
- しかし、攻撃を受けたかどうかを判断するのは難しい

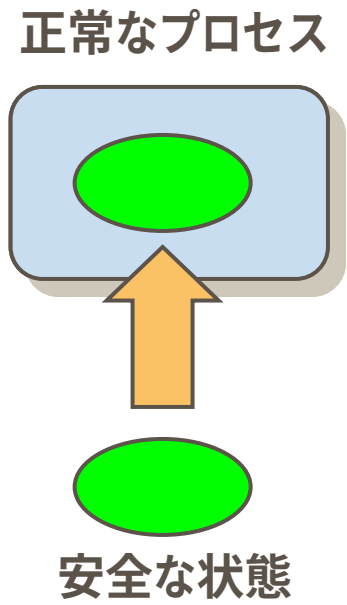


従来のアクセス制御機構

- Janus [Goldberg et al.'96] など
  - 動的な変更は許さない
- SubDomain [Cowan et al.'00]
  - 変更には秘密鍵が必要なので安全に見える
  - 乗っ取られると秘密鍵も知られてしまう
- Capabilityシステム [Tanenbaum et al.'85など]
  - capabilityを持っている間だけ権限を使える
  - 乗っ取られると渡された権限を悪用される

プロセスクリーニング

- アクセス制限を変更する前にプロセスを攻撃される前の状態に戻す
  - **プロセスの制御**を取り戻す
    - 乗っ取られたままアクセス制限を変更させない
  - **メモリイメージ**を元に戻す
    - トロイの木馬をメモリに残したままアクセス制限を変更させない



プロセスクリーニングの実装

- save_stateシステムコール
  - プロセスの状態を保存する
    - レジスタ、メモリイメージなど
    - カーネル内に保存
- restore_stateシステムコール
  - 保存しておいた状態を復元する
  - save_stateのあとに適用されたアクセス制限を解除する

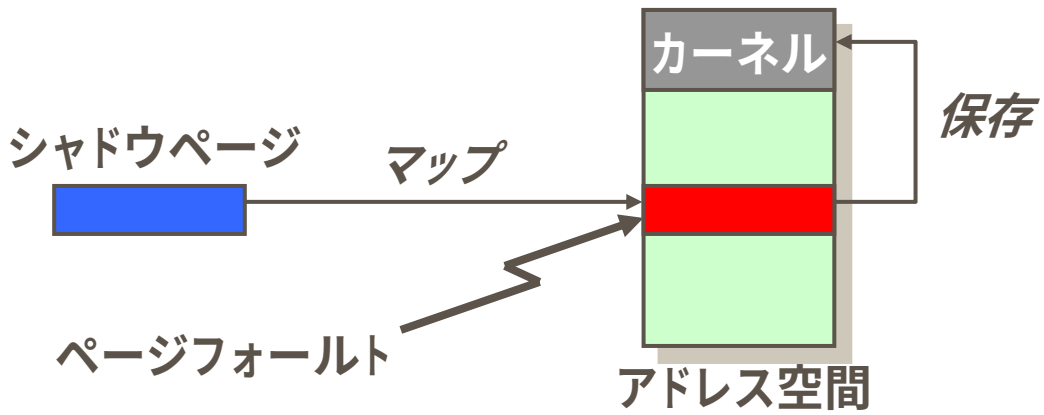
```
save_state();
accept();

＜アクセス制限＞
＜リクエスト処理＞

restore_state();
```

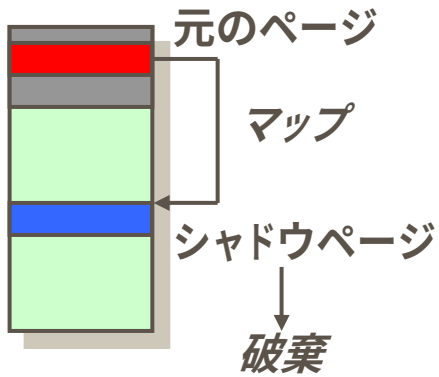
メモリイメージの保存の遅延

- copy-on-write技術により変更されたページだけ
  - save_stateで全てのページを書き込み禁止にする
  - ページフォールトが起こったらシャドウページと置き換える
    - 保存するページの内容をシャドウページにコピーする

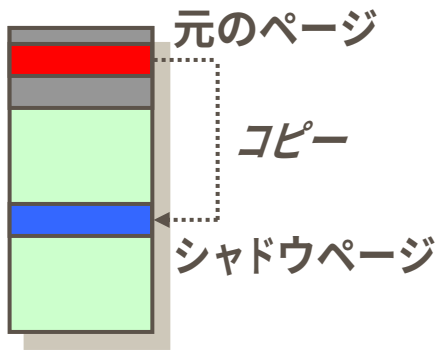


メモリイメージの復元方式

- 再マップ方式
  - 保存しておいた元のページをマップ
  - 次のページフォールトで再び保存される

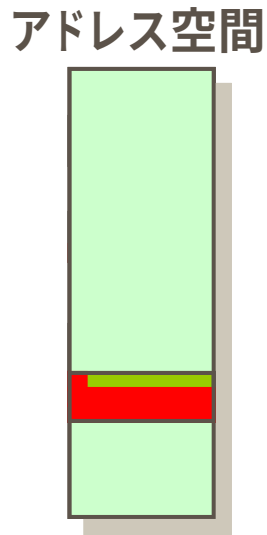


- コピー方式
  - 元のページの内容を**投機的に**シャドウページにコピー
  - 復元後に同じページが使われた時ページフォールトが起こらない
    - 使われなければコピーが無駄になる



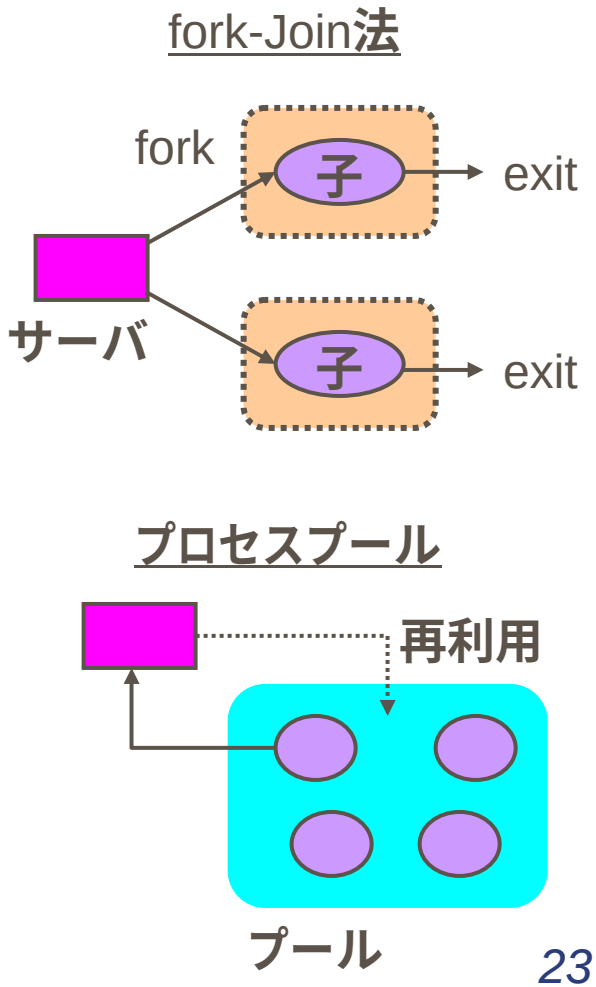
静的データのレイアウトの最適化

- 変更されるメモリページ数が最小になるようにサーバプログラムの静的データを再配置
  - ライブラリを静的にリンクする
    - データ・BSSセグメントを一か所にまとめる
  - 実行時プロファイルに従って静的データを並び替える
    - 変更されるデータをなるべく同じページに配置する



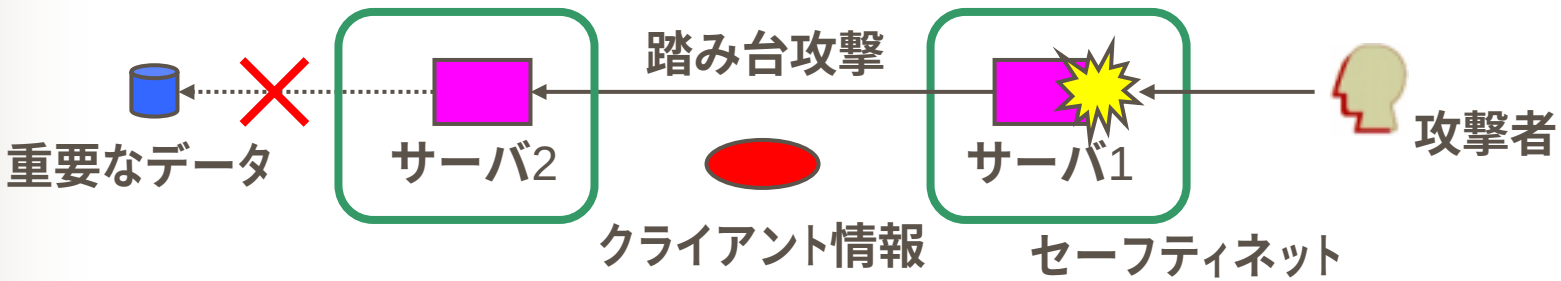
Fork-Join法との比較

- Fork-Join法でもプロセスクリーニングと同様の効果を得られる
  - 子プロセスにアクセス制限をかけて処理させる
  - セキュリティ
    - 子プロセスを作る時には親プロセスの状態がコピーされるので同等
  - 性能
    - オーバヘッドが大きい
    - プロセスプールが使えない



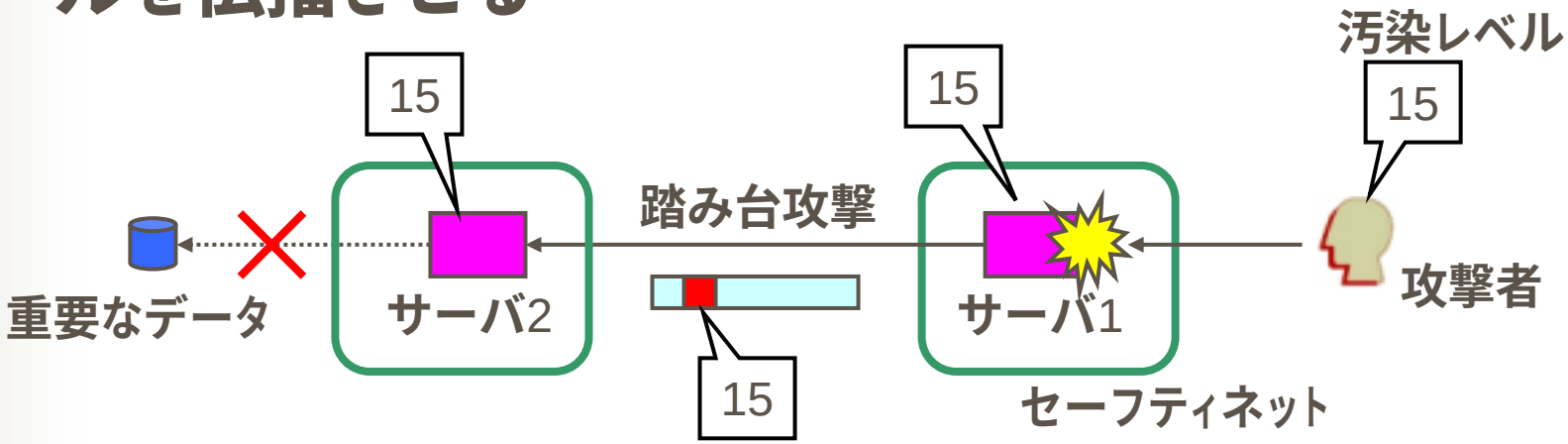
クライアントの行動の追跡

- 直接の通信相手のIPアドレスだけでは、攻撃者を判別することができない
  - 踏み台攻撃を防げない
- クライアント情報を記録し、実行フローに従ってその情報を伝播させる
  - 攻撃先のサーバにもセーフティネットを構築できる



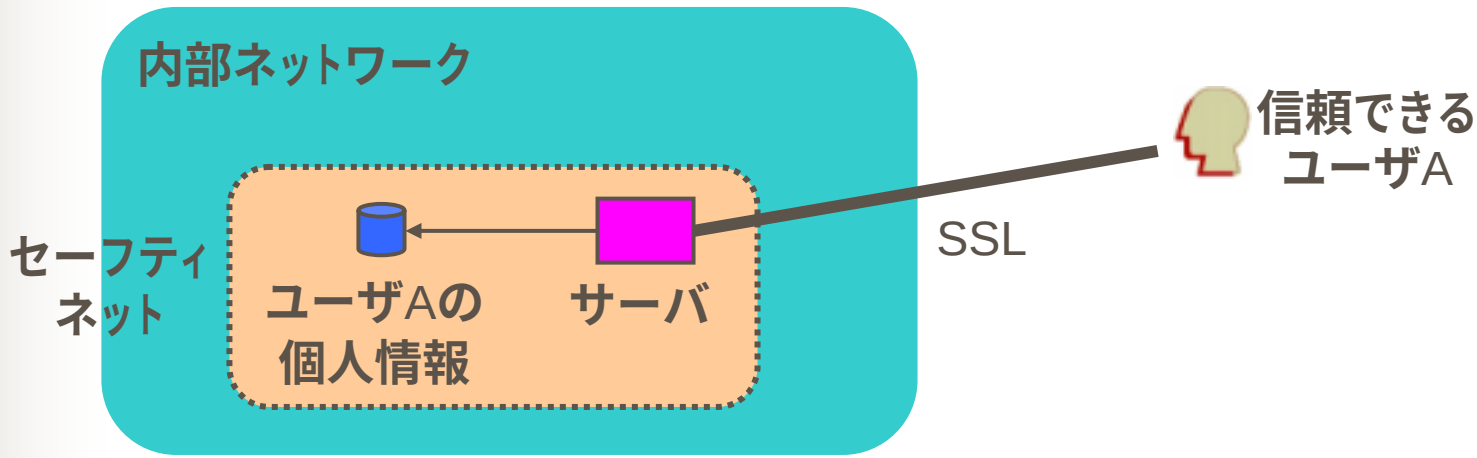
クライアント情報の圧縮

- クライアント情報は**汚染レベル**というパラメータに圧縮する
  - 実行フローに含まれるホストの最大の危険度
- パケットのIPオプションを使ってホスト間で汚染レルを伝播させる



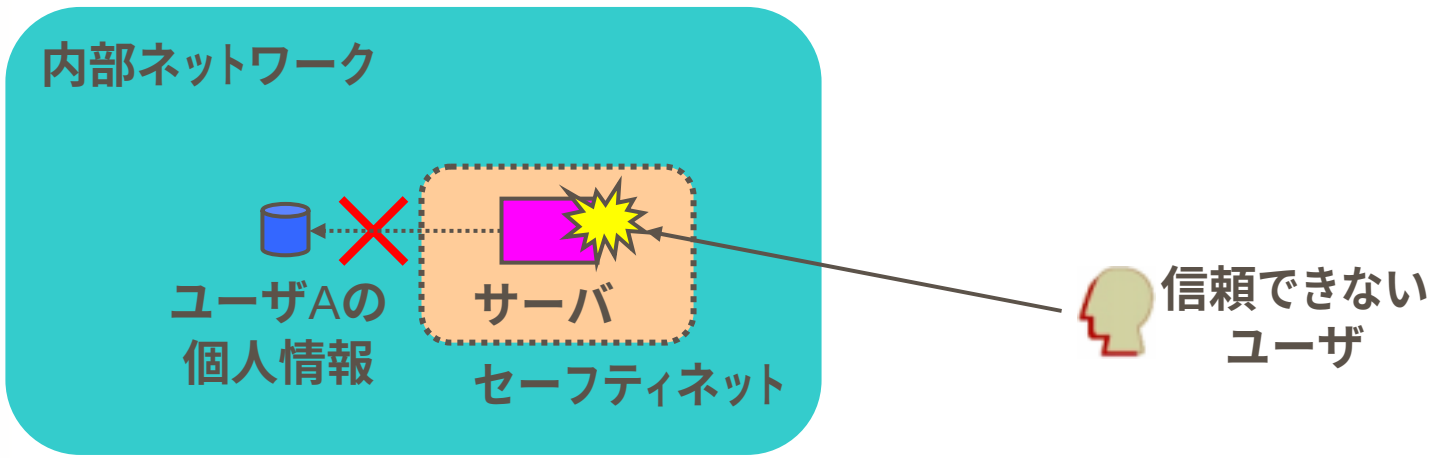
ネットワークレベルのユーザ認証

- サーバのアクセス制限に外部ホストの信頼できないIPアドレスは使えない
  - しかし、外部からはアクセスできないのでは不便
- 外部からのアクセスはユーザ認証に基づき制限
  - カーネルが自動的にSSLの認証を行う



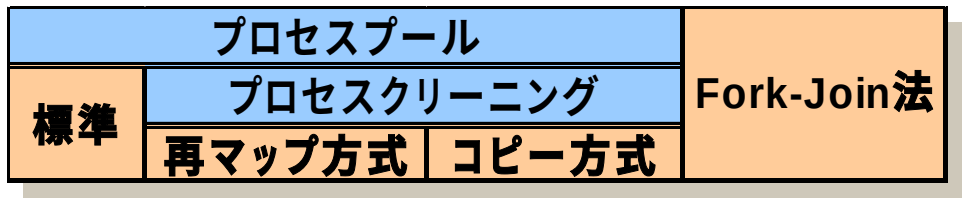
ネットワークレベルのユーザ認証

- サーバのアクセス制限に外部ホストの信頼できないIPアドレスは使えない
  - しかし、外部からはアクセスできないのでは不便
- 外部からのアクセスはユーザ認証に基づき制限
  - カーネルが自動的にSSLの認証を行う



実験

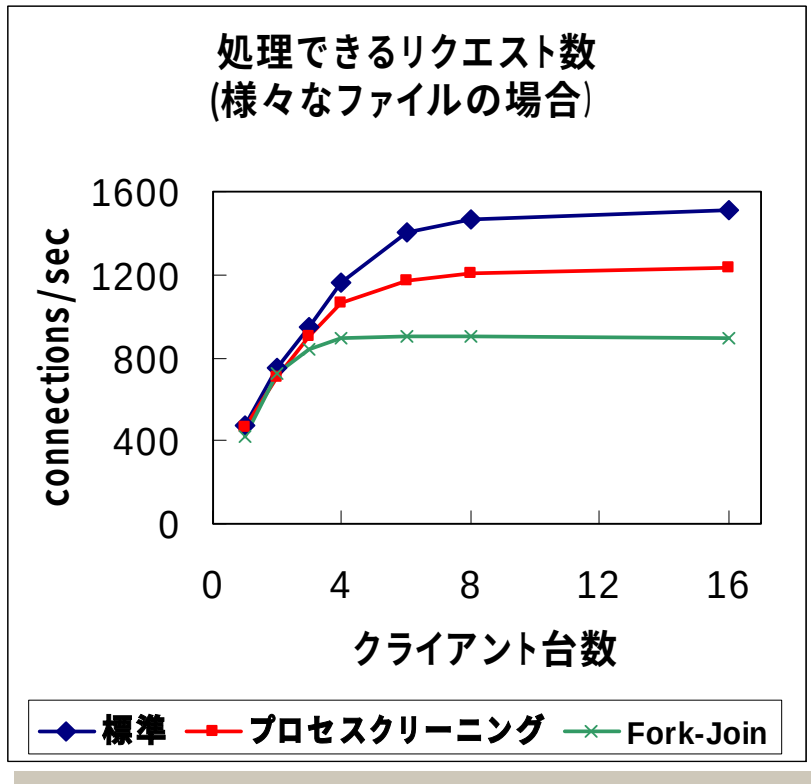
- Apacheウェブサーバの性能を測定
  - 0バイトの同じファイルをリクエストした時の性能
  - 様々なファイルをランダムにリクエストした時の性能



- 実験環境
  - サーバ：PentiumIII 933MHz, Compacto OS
  - クライアント：Celeron 300MHz, FreeBSD 3.4 16台
  - 100Mbpsイーサネット
  - WebStoneベンチマーク

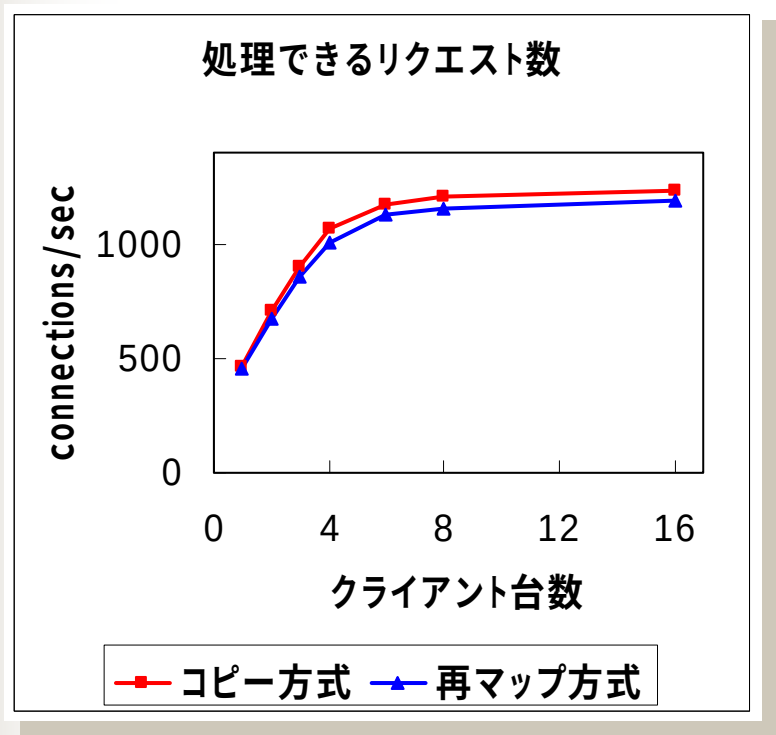
実験結果

- プロセスクリーニングのオーバヘッド (最大)
  - 0バイトファイル：36%
  - 様々なファイル：18%
- Fork-Join法に比べた性能向上率
  - 0バイトファイル：60%
  - 様々なファイル：30%

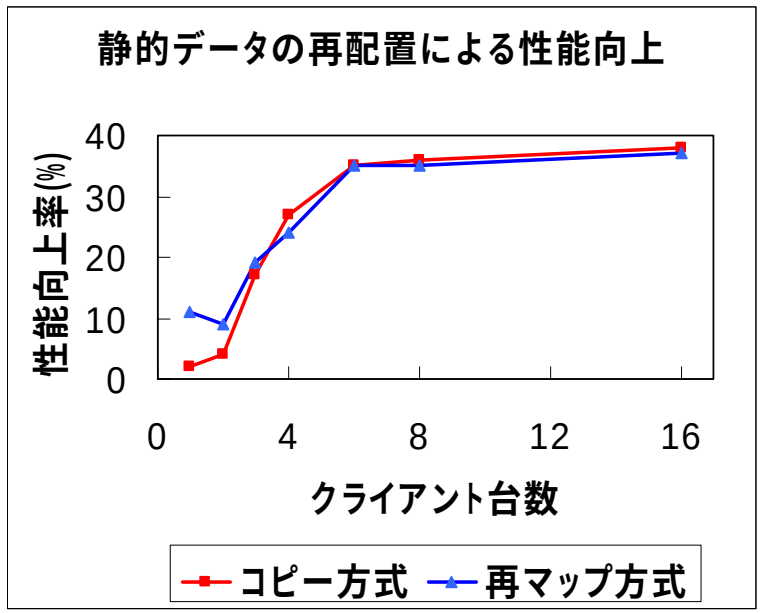


最適化の効果

- コピー方式は再マップ方式より5%速い



- 静的データの再配置により40%近く性能が改善する



第Ⅰ部のまとめ

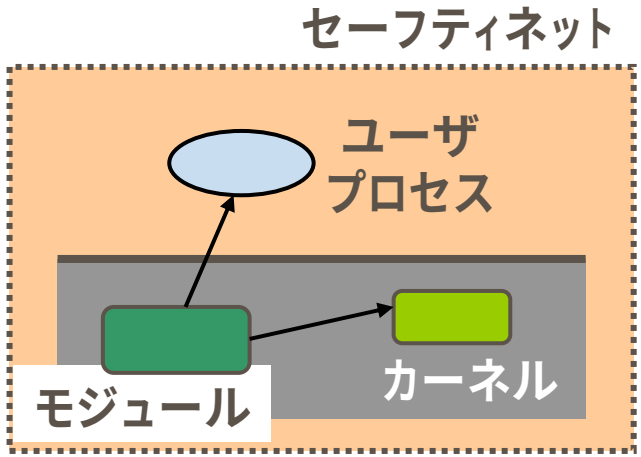
- ユーザレベルサーバ用のアクセス制御機構を開発した
  - プロセスクリーニングを提案した
    - 安全にアクセス制限を変更できる
  - クライアントに応じたセーフティネットを実現した
    - クライアントを正しく認識する技術の開発
- プロセスクリーニングの性能を測定した
  - オーバヘッドは最大で36%
  - 従来のFork-Join法より30〜60%高速

（II）OSモジュール用の保護機構

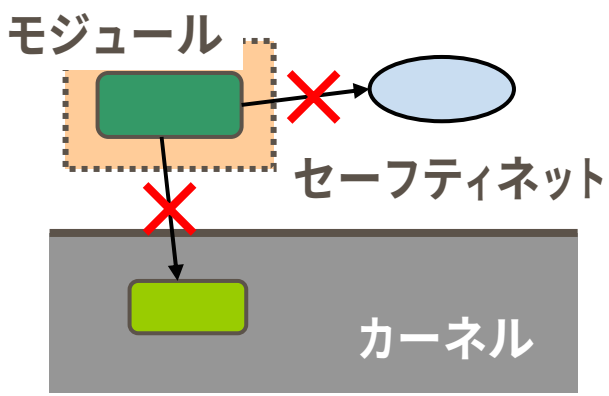
～マルチレベル・プロテクション～

OSモジュール用の従来のセーフティネット

- LKM（ロード可能カーネルモジュール）は範囲が広すぎる
  - 性能はよいが危険



- Mach [Accetta et al.’86]は範囲が狭すぎる
  - 安全だが性能が悪い

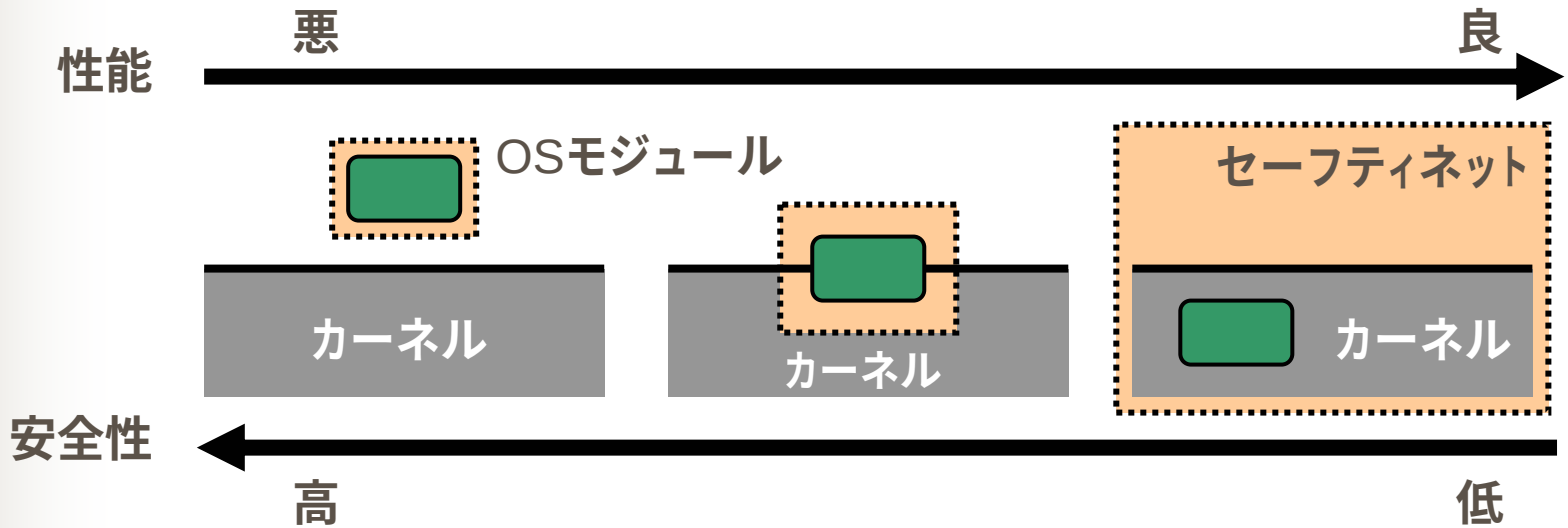


性能と安全性のトレードオフ

- OSモジュールの安定度は様々であるのに、セーフティネットは固定であるのが問題
  - ユーザが性能と安全性のトレードオフを変更できるようにすればよい
- 解決すべき問題
  - 変更はできるだけ簡単にできるようにすべき
  - 性能を最優先にした時にオーバヘッドができるだけ小さくなるようにすべき
  - 柔軟なトレードオフがとれるようにすべき

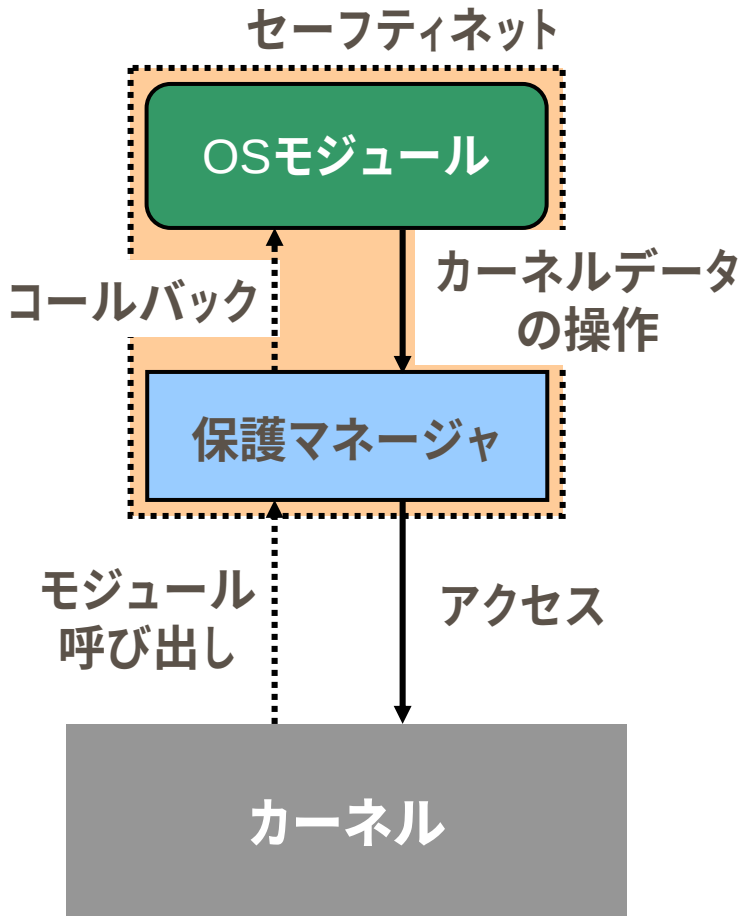
マルチレベル・プロテクション

- OSモジュールを様々な保護レベルでOSに組み込むことができる
  - 安定度に応じて性能と安全性のトレードオフを取ることができる



保護マネージャ

- OSモジュールとカーネルの間のゲートウェイ
  - カーネルからの呼び出しを中継する
  - OSモジュールにカーネルデータを安全に操作させる
- OSモジュールから他の部分を保護する

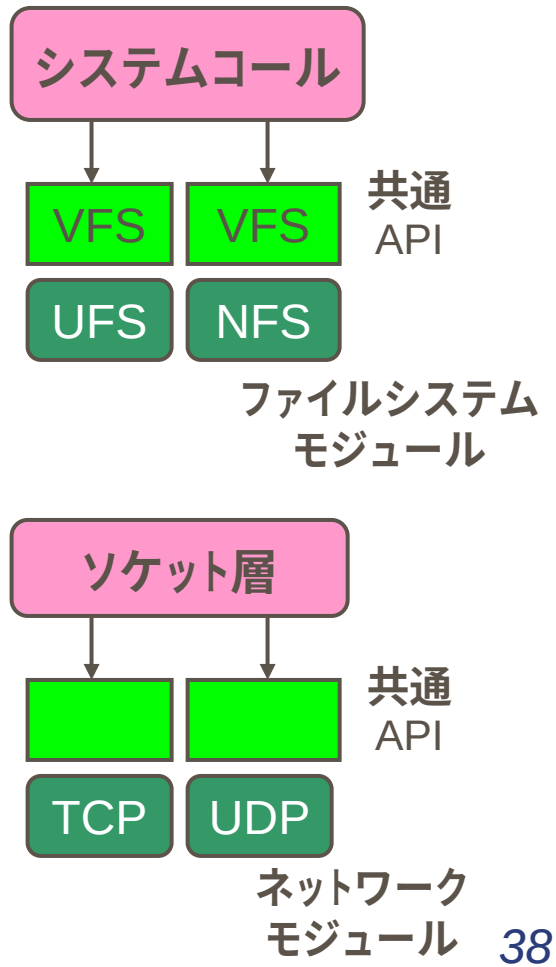


保護レベルの変更

- 保護マネージャの交換によってOSモジュールの保護レベルを変更する
  - 保護マネージャは複数用意されている
    - それぞれが異なる保護レベルを実現している
  - OSモジュールのソースコードを変更したり、コンパイルしたりする必要はない
    - 保護マネージャの実体はライブラリ
      - 再リンクのみでよい

共通APIの提供

- 全ての保護マネージャが共通のAPIを提供し、交換を容易にする
  - それぞれの保護マネージャの実装方法の違いを吸収する
- APIの種類
  - コールバック用のAPI
    - VFSのインタフェース
    - トランスポート層のインタフェース
  - カーネルデータ操作用のAPI
    - 抽象度の高いデータ構造を見せる

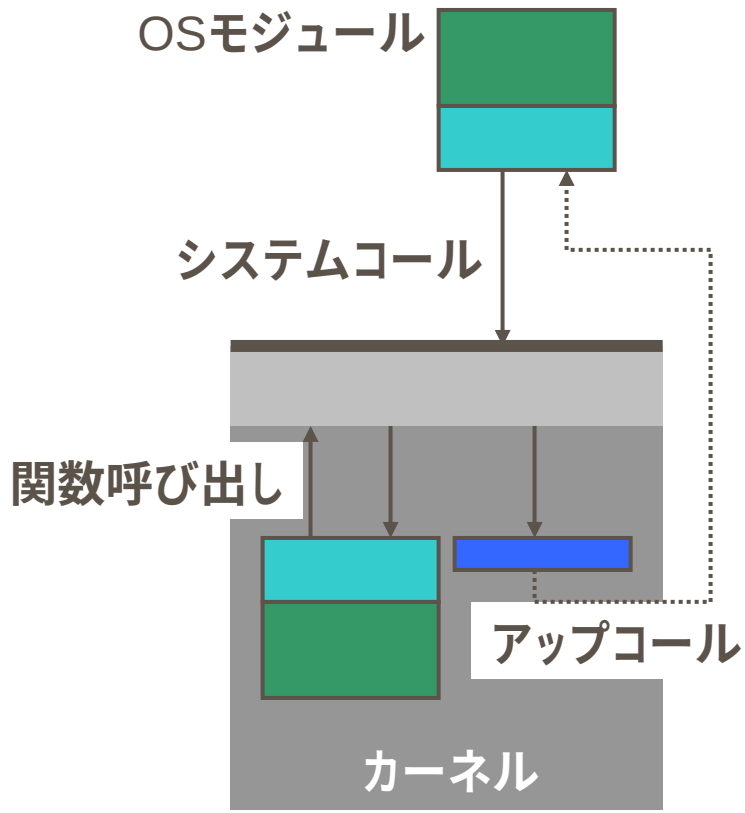


様々な保護レベルの実現

- 保護マネージャは以下の保護技術を組み合わせて様々な保護レベルを提供する
  - 不正メモリアクセスの検出
    - OSモジュールをユーザ空間に配置する
    - カーネルデータを保護する
    - カーネルデータを複製して内容を検査する
  - デッドロックの検出
    - wait-for-graphを作成して定期的に検査する

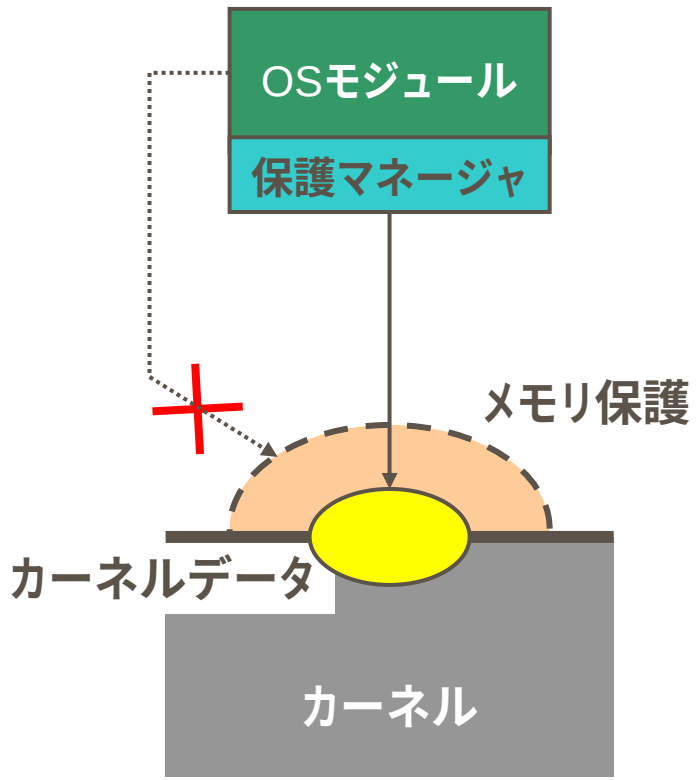
アドレス空間の保護

- OSモジュールをユーザ空間に配置する
  - アップコールで呼び出す
  - システムコールでカーネルの機能を利用
- OSモジュールがカーネルに影響を与えないようにする



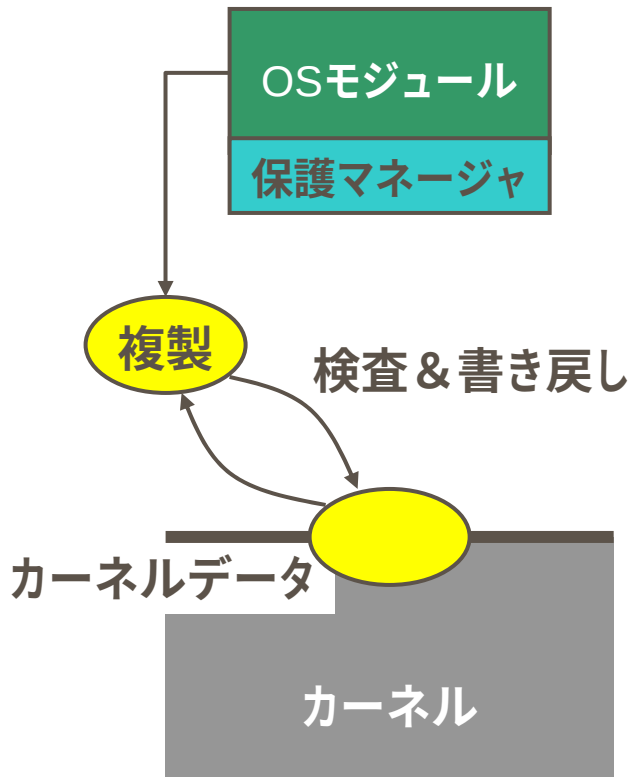
カーネルデータの保護

- カーネルデータの置かれる共有メモリを保護する
  - OSモジュールに制御がある間はmprotectで保護
  - 保護マネージャに制御がある間はアクセスを許可
- OSモジュールがカーネルデータを破壊してしまうのを防ぐ



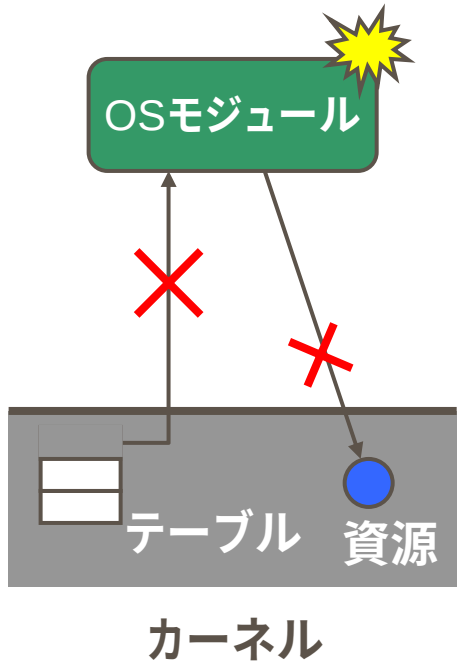
カーネルデータの複製

- カーネルデータを複製してそれにアクセスさせる
  - 保護マネージャが内容を検査してから書き戻す
- OSモジュールがカーネルデータの扱いを誤るのを防ぐ



エラーからの回復

- OSモジュールのエラーを検出したらカーネルからその影響を取り除く
  - 不正メモリアクセスを検出した場合
    - 以後そのモジュールを参照しないようにする
    - ログを元に安定な状態に戻す
  - デッドロックを検出した場合
    - wait-for-graphのループを破壊する



実験

- NFSサーバモジュールとTCPモジュールの性能を測定
- 表の5つの保護レベルとカーネルに直接作り込んだものについて実験した

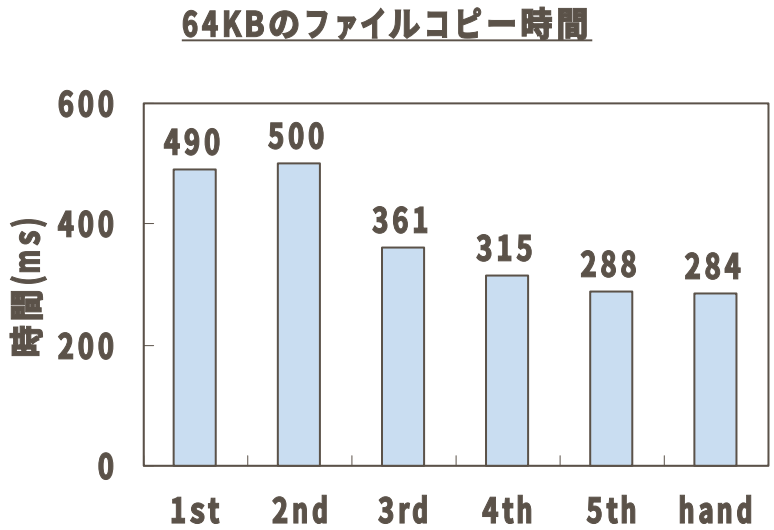
	1st	2nd	3rd	4th	5th
共有メモリの保護	○	△			
カーネルデータの複製	○	○	○		
アドレス空間の保護	○	○	○	○	

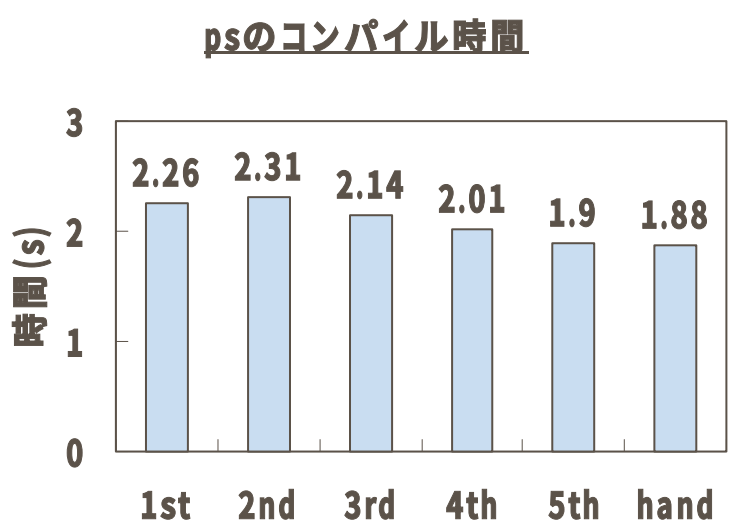
△：読み出し専用にして保護

- 実験環境
  - ＰＣ (PentiumII 400MHz, CAPELA OS) 2台
  - 10Mbpsイーサネット

NFSサーバモジュールにおける実験結果

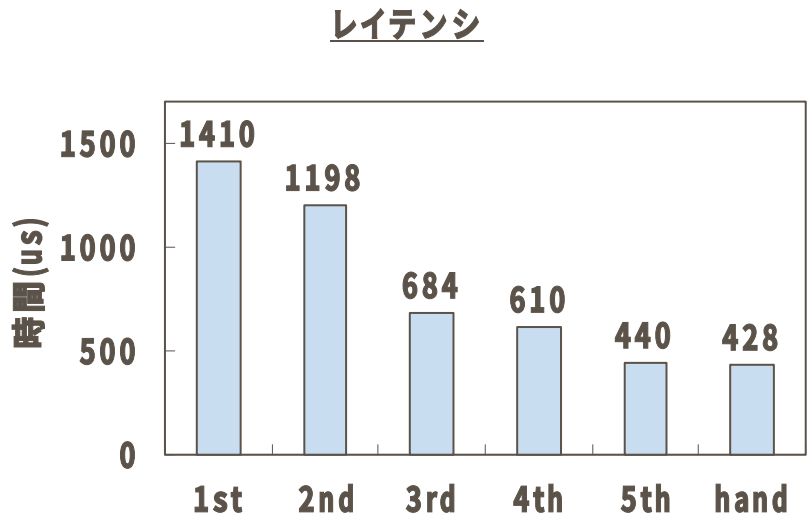
- 最高の保護レベルでのオーバヘッド
  - コピー：70%
  - コンパイル：20%
- 最低の保護レベルでのオーバヘッド
  - コピー：1.4%
  - コンパイル：1.1%

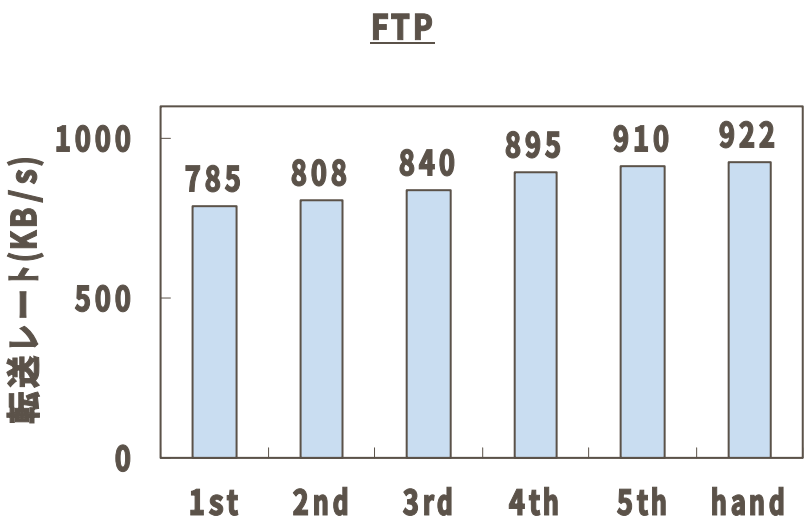




TCPモジュールにおける実験結果

- 最高の保護レベルでのオーバヘッド
  - レイテンシ：220%
  - FTP：16%
- 最低の保護レベルでのオーバヘッド
  - レイテンシ：2.8%
  - FTP：1.3%





関連研究

- Chorus [Rozier et al.’88]
  - **ユーザ空間とカーネル空間のどちらかに透過的にモジュールを配置できる**
  - **カーネル空間に配置した時のオーバヘッドが大きい**
- VINO [Seltzer et al.’94]
  - SFI[Wahbe et al.’93]**によりカーネル内モジュールの不正メモリアクセスを検出する**
- SPIN [Bershad et al.’95]
  - Modula-3**で記述することにより不正メモリアクセスを回避する**

第II部のまとめ

- OSモジュール用の保護機構を開発した
  - マルチレベル・プロテクションを提案した
    - 性能と安全性のトレードオフをとる
- 実験によりマルチレベル・プロテクションの有用性を確かめた
  - 保護レベルを低くすると性能が良くなった
  - 保護レベルを最低にした時、作り込みのモジュールの性能に十分近くなった

本発表のまとめ

- 性能を十分に考慮してソフトウェアの安全性を高めることを目標とした研究を行った
  - サーバソフトウェアに対して**動的なセーフティネット**を構築した
    - **プロセスクリーニング**によりサーバのアクセス制限を安全に変更できる
    - **マルチレベル・プロテクション**によりOSモジュールの性能と安全性のトレードオフをとれる

今後の展望

- マルチスレッドサーバへのプロセスクリーニングの適用
  - 安全性を保つには保護つきのスレッドが必要
    - メモリの保存･復元に関してはプロセスの場合と大差ない可能性が高い
- マルチレベル･プロテクションにおいて保護レベルを変更するガイドラインの作成
  - 統計を指標として利用したい
    - OSモジュールの使用年数、使用頻度、開発者のスキルなど

      対するバグの減り方を調べる必要がある

既発表論文

- プロセスクリーニング
  - **“サーバのアクセス制限を安全に変更するための機構”, *情報処理学会論文誌, 42巻6号***
  - “A Secure Access Control Mechanism against Internet Crackers”, *IEEE ICDCS’01 (short)*
- マルチレベル・プロテクション
  - **“多段階保護機構：拡張可能OSの新しいFail-safe機構”, *情報処理学会論文誌, 39巻11号***
  - “Operating System Support for Easy Development of Distributed File Systems”, *IASTED PDCS’98 (short)*