メモリ管理の常識を覆す：30年前に予言された「単一アドレス空間」が今、再評価される理由

1. 導入：OSの「当たり前」という名の「ベルリンの壁」を疑う

私たちが日常的に利用しているLinuxやWindowsといった現代のオペレーティングシステム（OS）には、一つの揺るぎない「鉄則」があります。それは、プロセスごとに独立したプライベートな仮想メモリ空間を割り当てる「マルチアドレス空間」モデルです。しかし、このモデルは決してコンピュータ・アーキテクチャにおける唯一の正解ではありません。実は、1960年代から続くこの仕組みは、32ビット時代までの「限られたメモリ資源（最大4GB）」を各プロセスで奪い合うための、いわば「狭小アパート」をやりくりするような歴史的な妥協案に過ぎません。プロセスごとに壁を築くことで、住所（名前空間）の枯渇を防いできたのです。ところが、64ビット時代の到来により、この前提は根底から覆されました。メモリ空間はもはや節約すべき希少資源ではなく、目の前に広がる「無限の広野」へと姿を変えたのです。現代のOSがプロセス間に築いている「隔離の壁」は、かつては必要悪でしたが、今やデータの移動を阻む「ベルリンの壁」となり、システムの効率を著しく損なっています。このパラダイムシフトを背景に、30年前から予言され、今再び熱い視線を浴びているのが「単一アドレス空間（SASOS）」という概念です。

2. 驚愕のスケール：64ビット空間は「5,000年」使い切れない

SASOS（Single Address Space Operating System）とは、カーネルや全プロセス、さらには一時的なデータから永続的なデータまで、システム上のあらゆるエンティティが「グローバルで共有される単一の仮想アドレス空間」を指し示すアーキテクチャです。この空間がいかに広大か、具体的な数値で考えてみましょう。完全な64ビット空間であれば、 **毎秒100MBの速度で新しいメモリ領域を消費し続けたとしても、すべてを使い切るまでに5,000年を要します。** 私たちが一生かかっても使い切れないほどの広大な「名前（アドレス）」が、すでに手元にあるのです。SASOSの核心は、メモリアドレスの「翻訳（名前解決）」と「保護（アクセス権）」を完全に切り離す（直交化する）点にあります。ここで一つの比喩を使ってみましょう。従来のOSは、鍵のかかった「個室」の集まりです。部屋の鍵を持っていない限り、中にどんな本があるかさえ分かりません。対してSASOSは、巨大な「一つの図書館」です。誰でもすべての本の背表紙（アドレス）を眺めることができます。しかし、実際に本を開いて中身を読むには、その本専用の「閲覧許可証（ケーパビリティ）」が必要になります。「アドレスを知っていること」と「アクセスできること」を分離することで、同じ空間に全員が同居しながら、厳格なセキュリティを維持できるのです。

3. 「ファイル」という概念の終焉：単一レベルストア（SLS）の衝撃

SASOSがもたらす最も革命的な変化が、「単一レベルストア（SLS: Single-Level Store）」です。これは、メインメモリ（RAM）とストレージ（SSD/HDD）の境界線を完全に消滅させます。従来のOSでは、ストレージのデータを使うために read() や write() といったシステムコールを呼び出し、メモリ上へ「翻訳コピー」を作る必要がありました。しかしSASOSでは、すべてのデータが仮想アドレス空間のどこかに一意にマッピングされています。これにより、開発者はポインタを用いたメモリアクセスだけで、ストレージ上の永続データに直接触れることが可能になります。ソース資料で述べられている「 **直交的永続性（Orthogonal Persistence）** 」とは、オブジェクトのライフタイムを意識せず、メモリ上のデータ構造をそのままの形で永続化できることを意味します。「ファイルに保存する」とい

う翻訳作業から解放された世界は、ソフトウェア開発の常識を根底から書き換えてしまいます。

4. 爆速の秘密：コンテキストスイッチの「フラッシュ」を捨てる

従来のOSでプロセスを切り替える際、CPUは「TLB（翻訳ルックアサイドバッファ）」という高速キャッシュをリセット（フラッシュ）しなければなりませんでした。プロセスごとにアドレスの意味が異なるため、情報を残すと他人のデータを誤参照してしまうからです。この「フラッシュ」に伴うオーバーヘッドが、システム全体のパフォーマンスを著しく低下させる要因となっていました。しかし、全プロセスが同じアドレス空間を共有するSASOSでは、TLBをフラッシュする必要がありません。保護ドメインの権限を切り替えるだけで済むため、プロセス間通信（IPC）やプロセス生成のコストが劇的に抑えられます。以下は、ソフトウェアベースのSASOSである「Singularity」と、従来型OSの性能を比較した驚異的なデータです。| OS名 (アーキテクチャ) | API呼び出し (サイクル) | スレッド切り替え (サイクル) | IPC Ping/Pong (サイクル) | プロセス生成 (サイクル) || ------ | ------ | ------ | ------ | ------ || **Singularity (SASOS)** | **80** | **365** | **1,040** | **388,000** || FreeBSD | 878 | 911 | 13,300 | 1,030,000 || Linux | 437 | 906 | 5,800 | 719,000 || Windows | 627 | 753 | 6,340 | 5,380,000 | CADツールや大規模データベースのように、膨大なデータを頻繁に共有・参照するシステムにおいて、データをコピーせず「ポインタを渡すだけ」で完了するゼロコピーの効率性は、圧倒的な優位性をもたらします。

5. 負の遺産：なぜ「1970年代のUNIXの幽霊」に勝てなかったのか

これほど優れたSASOSが普及しなかった背景には、歴史的な高い壁がありました。最大の障害は、POSIX規格の fork() システムコールとの根本的な矛盾です。fork() は「親の空間を丸ごとコピーして別のアドレス空間を作る」という動作を前提としています。しかし、単一アドレス空間では「同じアドレスに別のデータを入れる」ことは許されません。子プロセスのデータを別の場所にずらしてコピーすれば、プログラム内の「絶対ポインタ」が古い場所を指したままになり、確実にクラッシュします。さらに「共有ライブラリ」の問題も立ちはだかりました。標準ライブラリ（libcなど）が持つ静的変数は、プロセスごとに独立した値を持つ必要があります。しかし、全員が同じアドレス空間を共有すると、一人が書き換えた値が全員に影響してしまいます。これを防ぐには「グローバルオフセットテーブル（GOT）」のような複雑な間接参照が必要になり、SASOSの「直感的で一意なアドレス」という利点が損なわれてしまったのです。CやC++で書かれた膨大なレガシー資産という「負の遺産」を移行するコストは、あまりにも天文学的でした。

6. ルネサンス：WebAssembly、CHERI、そして「生ける化石」の逆襲

長らく眠っていたSASOSの思想は今、ハードウェアとソフトウェアの両面から劇的な復活を遂げようとしています。まず注目すべきは、商用環境での唯一の成功例である「**IBM i（旧 OS/400）**」です。このシステムは128ビットの太いポインタを採用し、単一レベルストアを実現しています。驚くべきは、TIMIと呼ばれる中間層により、1980年代の48ビットアーキテクチャ向けバイナリが、現代の64ビットPowerPC上で再コンパイルなしに動作し続けていることです。SASOSが単なる夢物語ではないことを、この「生ける化石」は証明し続けています。現代の技術も、この背中を追っています。

- **WebAssembly（Wasm）：** 単一のホストプロセス内で多数のモジュールを軽量に隔離して実行する仕組みは、SASOSの思想そのものです。
- **CHERI（シェリ）：** ポインタに権限情報を付与するハードウェア拡張により、単一空間内での「細粒度な隔離」を高速に実現します。

- **μFork：** CHERIを応用し、fork() 時に絶対ポインタを自動で書き換えることで、**FreeBSDのforkより3.7倍も高速（わずか54μs）**なプロセス生成を実現し、POSIXの壁を突破しようとしています。SASOSは、決して過去の失敗作ではありません。資料の結論が示す通り、**「SASOSは『過去の失敗』ではなく、ハードウェア（CHERI等）や実行基盤（Wasm等）が追いつくのを待っていた『早すぎた正解』」**だったのです。

7. 結びに：計算資源の「真の自由」へ向けて

計算資源の極限的な効率と、厳格なセキュリティの両立が求められる今、私たちが慣れ親しんできた「OSの境界線」は再び書き換えられようとしています。もし、メモリとストレージの壁が消え、ファイル入出力という概念さえ過去のものになったら、あなたのソフトウェア設計はどう変わるでしょうか？私たちはいつまで、1970年代の制約が生んだ「コンテキストスイッチという名の重税」を払い続けるのでしょうか。境界線のない、真にフラットなコンピューティングの夜明けは、すぐそこまで来ています。