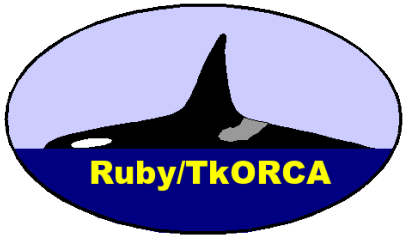
2005年下期未踏ソフトウェア創造事業
『ネット公開を目的としたマルチウィンドウアプリ用フレームワーク』

未踏プロジェクト Ruby/TkORCA が目指すもの

～ Ruby/TkORCA (Ruby/Tk On RFB Canvas) ～

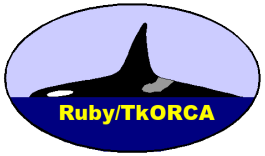
『GUI + ネット公開』というキーワードに興味を持つすべての人のために...



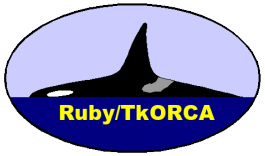
永井 秀利

九州工業大学 情報工学部 知能情報工学科

2006/01/28



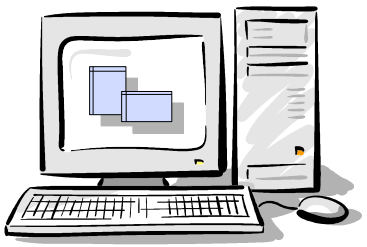
CONTENTS

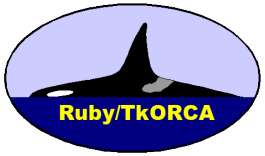


開発を目指すフレームワーク Ruby/TkORCA とは？

- ローカルのウィンドウシステムで稼働している
- マルチウィンドウアプリケーションを，
- 表示や操作性をそのままに，
- 再プログラミングの必要なしに，
- 簡単かつ安全に，
- 必要なら実行の監視や制約の機能を付加できて，
- サーバの負荷はできるだけ少なく抑えて，
- PDA 等の低能力端末を含む多種多様なクライアントで使えるように
- ネットワーク公開アプリケーション化する

ためのフレームワーク

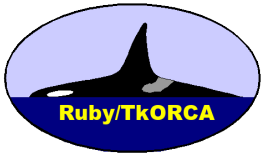




一言で言うなら．．．

どこでもGUI!!

（GUI, Anyware!!）

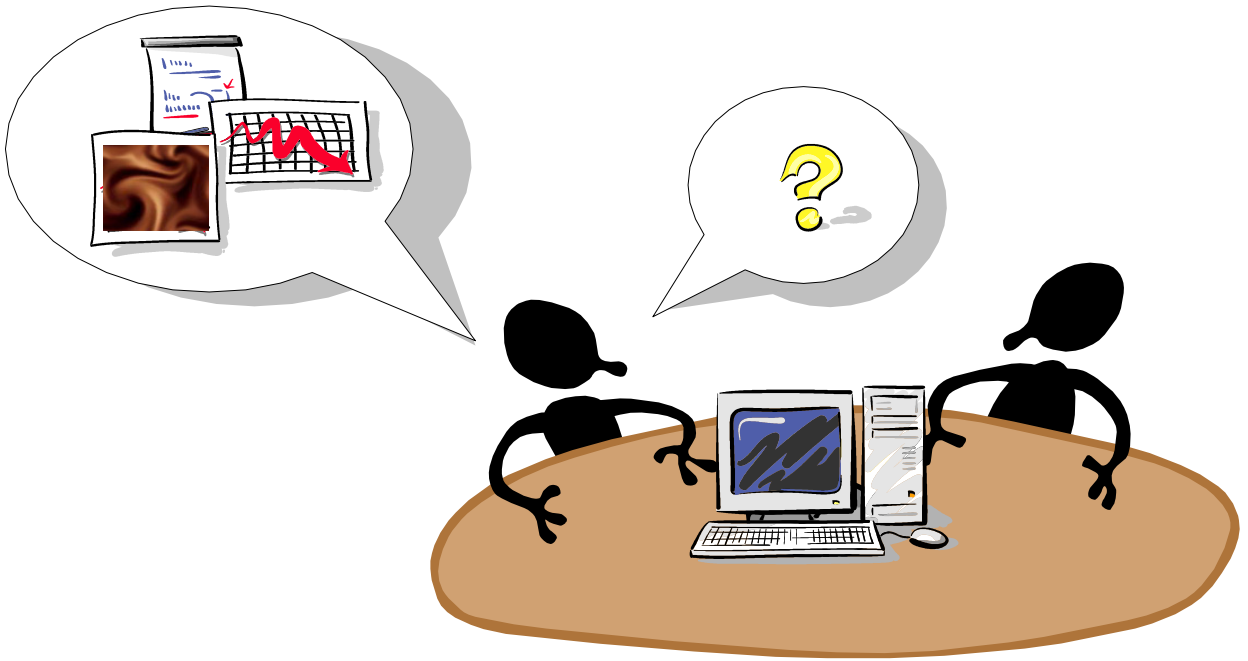


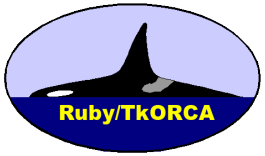
現在の状況では...

A: 『こんな GUI で研究データ分析用ツールを作ってみたけど，どうかな? 』

B: 『へぇ，いいねぇ．マルチウィンドウでグラフや可視化したデータが表示されるってわけか』

A: 『うん．いいライブラリを見付けたんでね．表示されてる可視化データもインタラクティブに操作できるよ』

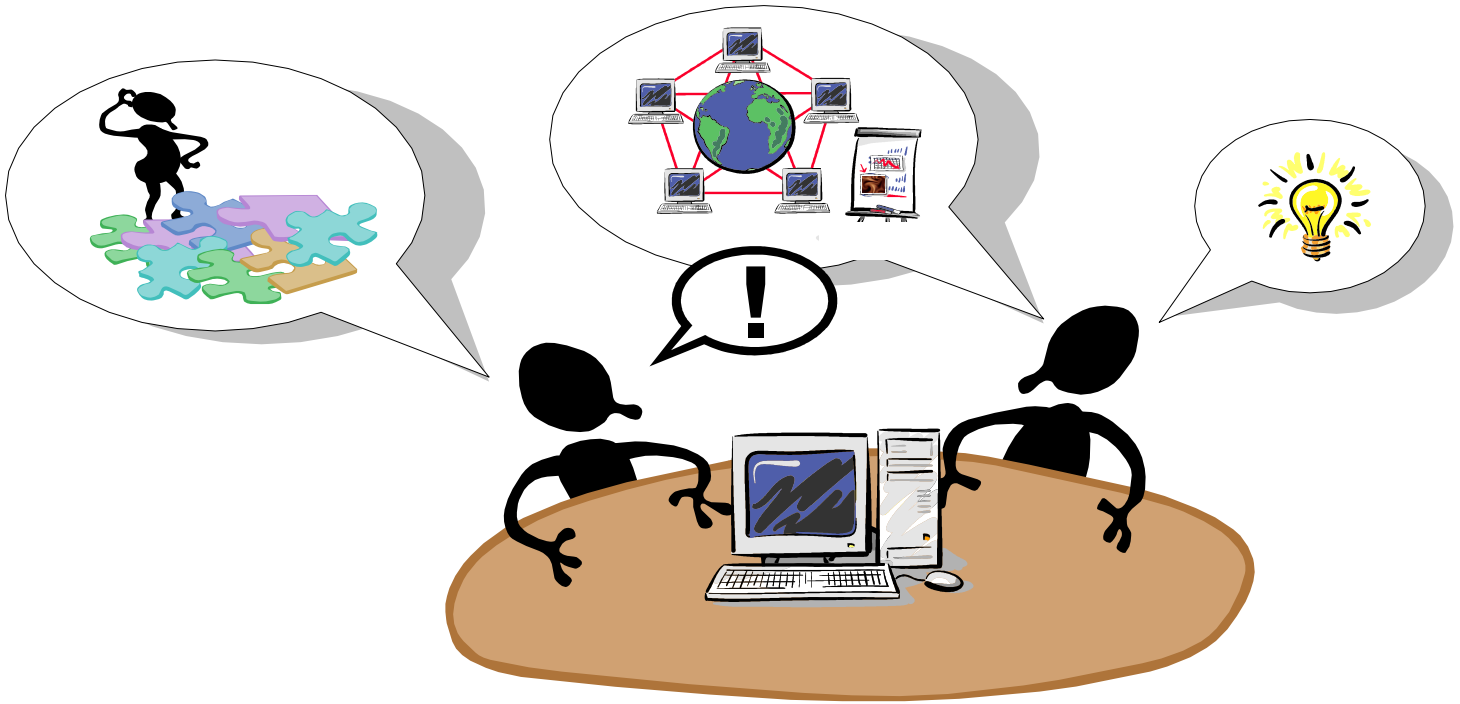


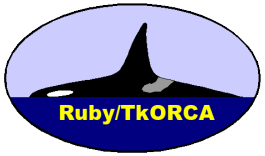


現在の状況では...

B: 『これってさぁ，デモシステムとして外部に公開できないかなぁ?
Ajax とかでやれば，何とかなるんじゃないの? 』

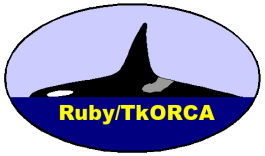
A: 『えっ? それじゃ頭から作り直すのと同じじゃん.
操作性も変わるし，せっかく見付けたライブラリも使えないし...
それに，研究上の重要情報の一部がクライアントに渡っちゃうかも』





現在のネットワークアプリにおける問題点

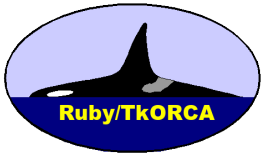
- ローカルの GUI との間での操作性のギャップ
- thick client アプローチでの操作性向上を目指すことによる問題増加
  (Ajax などによる他，独自プロトコルによる場合も含む)
  - サーバ側とクライアント側とのそれぞれを個々に開発するコストの増加
  - クライアントの差異に気をつかう必要性の増大
  - クライアントの能力に対しての要求や依存性の増大 (＝利用可能範囲の減少)
  - クライアント側プログラムの配布やインストールのコストの増加
  - クライアント依存の増大によるサーバ側プログラム更新の即応性の低下
  - クライアントに渡すプログラムやデータ量の増加による情報漏洩リスクの増大
  - そのままローカルでも使うには，要求される実行環境が不必要に重い
- 画面出力を行う有用なライブラリの利用が困難



現在の状況では...

B: 『そうかぁ... なら，このアプリを VNC で直接公開しちゃったら? 』

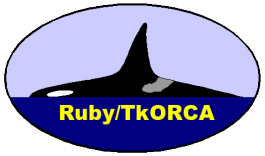
A: 『これ，セキュリティについては考えてないし，第一，サーバのウィンドウシステムを誰かもわかんない人に直接操作させる気? 』



現状での GUI アプリの外部公開における問題点

- X プロトコル
  - クライアントに対する制約が大きい上，プロトコル自体が重すぎる
- Windows リモートデスクトップ
  - マシンの遠隔操作用であり，特定アプリのみのネット公開目的には役に立たない
- VNC (RFB プロトコル)
  - 一般的な利用形態はマシンの遠隔操作 → 他の利用形態への想定や配慮が希薄
  - 運用上，ウィンドウマネージャが癌
    - かなり大きな操作権限を与えてしまうにもかかわらず，動作の監視ができない
    - 稼働プロセス数が増えるなど，結構重い上に起動も遅くなる
    - にもかかわらず， GUI 操作性維持には必須（Windows では置き換えも不可）

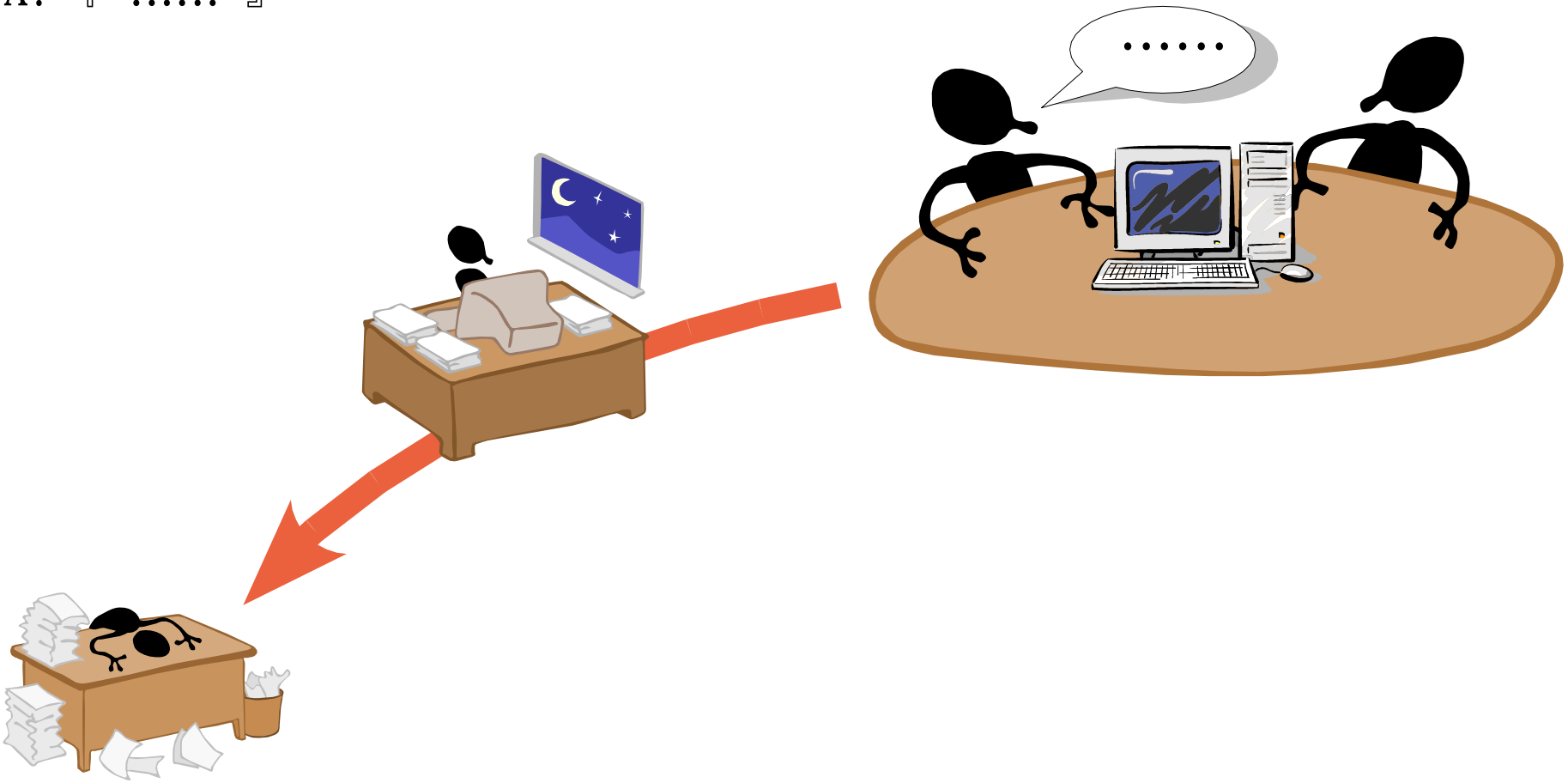
→ ローカルの個々の GUI アプリケーションのみを
広く安全に公開するための枠組の不在



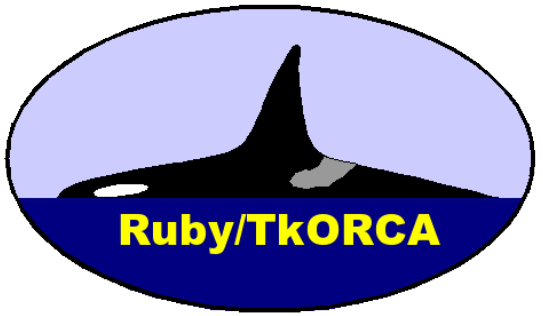
現在の状況では...

B: 『うーむ，どうにもならないか...
やっぱり公開用は別に作んなきゃいけないかなぁ...』

A: 『 ..... 』



Ruby/TkORCA が提供する枠組を活用すると…

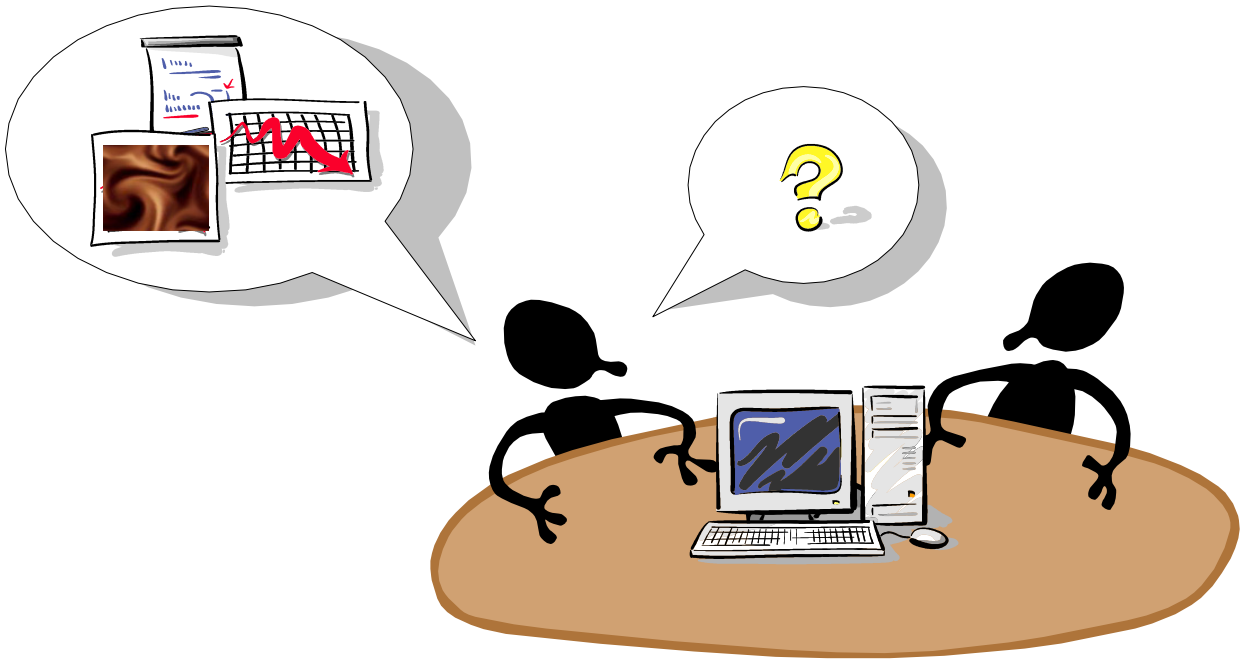


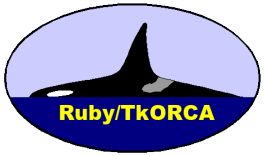
Ruby/TkORCA を活用すると ...

A: 『こんな GUI で研究データ分析用ツールを作ってみたけど，どうかな? 』

B: 『へぇ，いいねぇ．マルチウィンドウでグラフや可視化したデータが表示されるってわけか』

A: 『うん．いいライブラリを見付けたんでね．表示されてる可視化データもインタラクティブに操作できるよ』

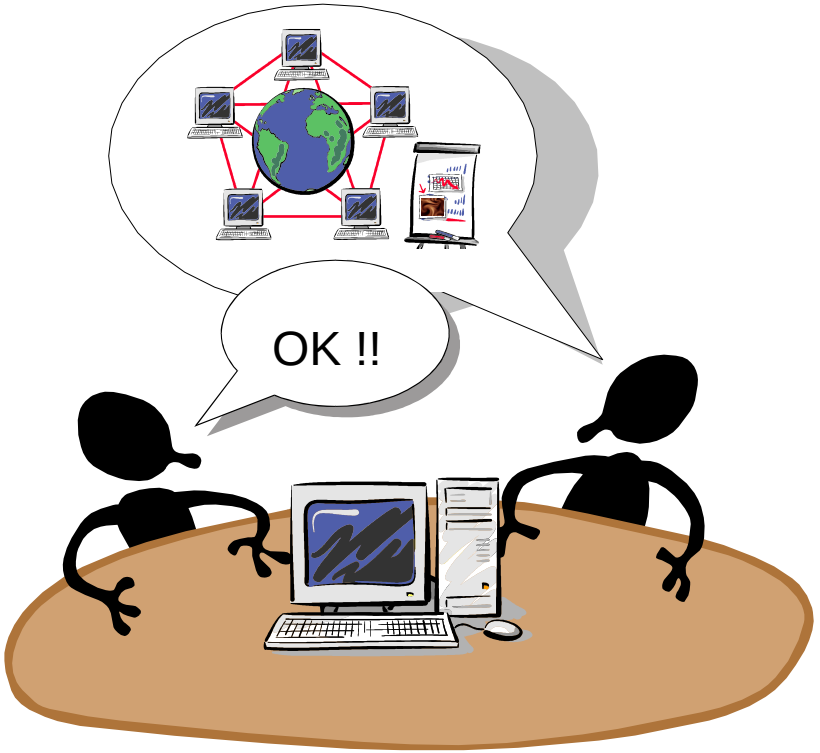


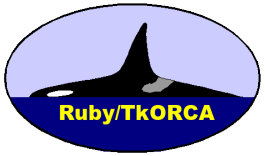


Ruby/TkORCA を活用すると ...

B: 『じゃあこれ，デモシステムとして外部に公開しようよ』

A: 『そうだね．では，早速テスト環境で実行してっと...』





本フレームワークを用いる利点(1)

- ローカルで動くGUIアプリケーションをそのまま利用可能
  - 公開専用のアプリケーションを新たに作成する必要なし
  - 重い環境が必要な公開アプリを我慢してローカルで使う必要なし
- ローカルのウィンドウシステムでの実行テストが可能
  - 実行テストのために特別なサーバを動かす必要なし
  - 公開対象アプリを公開時と全く同じ実行環境で実行可能
  - ローカルのウィンドウシステム上で，公開時と同じに表示や操作が可能

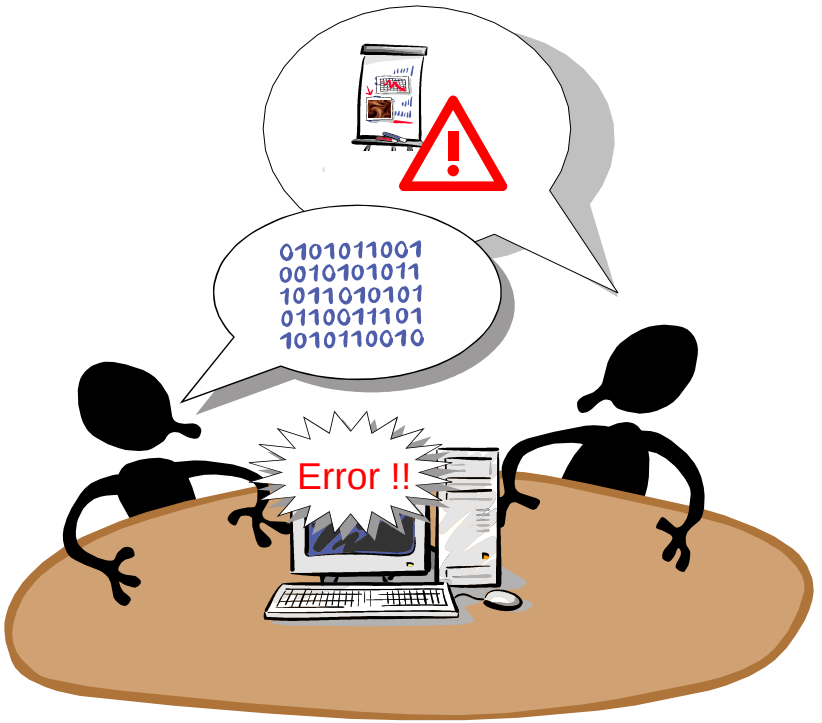
→ 開発の速度や効率の向上

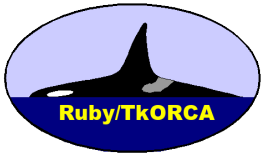
Ruby/TkORCA を活用すると ...

A: 『あ，セキュリティエラーか...
　　どうやら，データディレクトリにアクセスしている部分みたい』

B: 『その程度のエラーならソースはそのままで，指定ディレクトリへの
　　アクセス許可を監視スクリプトで設定すれば問題ないね』

A: 『うん，それで十分だと思う．
　　ではチェックを追加して... よし，再テストと...』





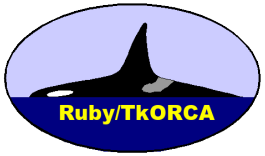
本フレームワークを用いる利点(2)

- プログラムもデータもクライアントには送らない

  → 低い情報漏洩リスク

- セキュリティ上の致命的問題がない限りはソース変更の必要なし
  - 公開対象アプリケーションは監視システムによる監視下の sandbox で実行
  - 公開前の実行テストに合格 = 最低限のセキュリティは確保
  - 一般的なセキュリティ上の制約や許可程度なら，監視システムの設定で対処可能
  - より複雑な監視や制御も，監視スクリプトを記述することで対応可能

  → セキュリティ確保を支援する機能により
  安全性向上に要するコストを低減



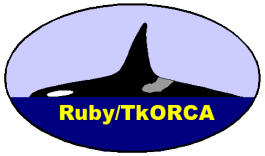
Ruby/TkORCA を活用すると ...

A: 『今度は大丈夫そうだね』

B: 『ならサーバに設定しよう．すぐ終わるからちょっと待って...
　どう？ 』

A: 『動いてる．大丈夫みたい．あとは宣伝するだけか』

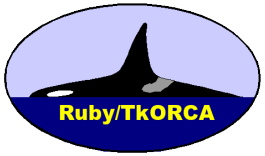
B: 『だね．アナウンスよろしく！』



本フレームワークを用いる利点(3)

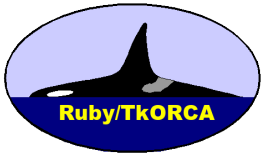
- 容易な公開設定と柔軟なクライアント対応
  - 公開時環境と同等のテスト環境で確認済みのため，公開や更新の作業はサーバでの登録を書き換える程度
  - thin client であるため，パソコンに限らずPDAクラスの機器でもクライアントになることが可能
  - クライアントへの依存性がほとんどないため，サーバ上のアプリケーションを更新してもクライアントの更新は必要なし

→ ニーズに対しての極めて高い即応性



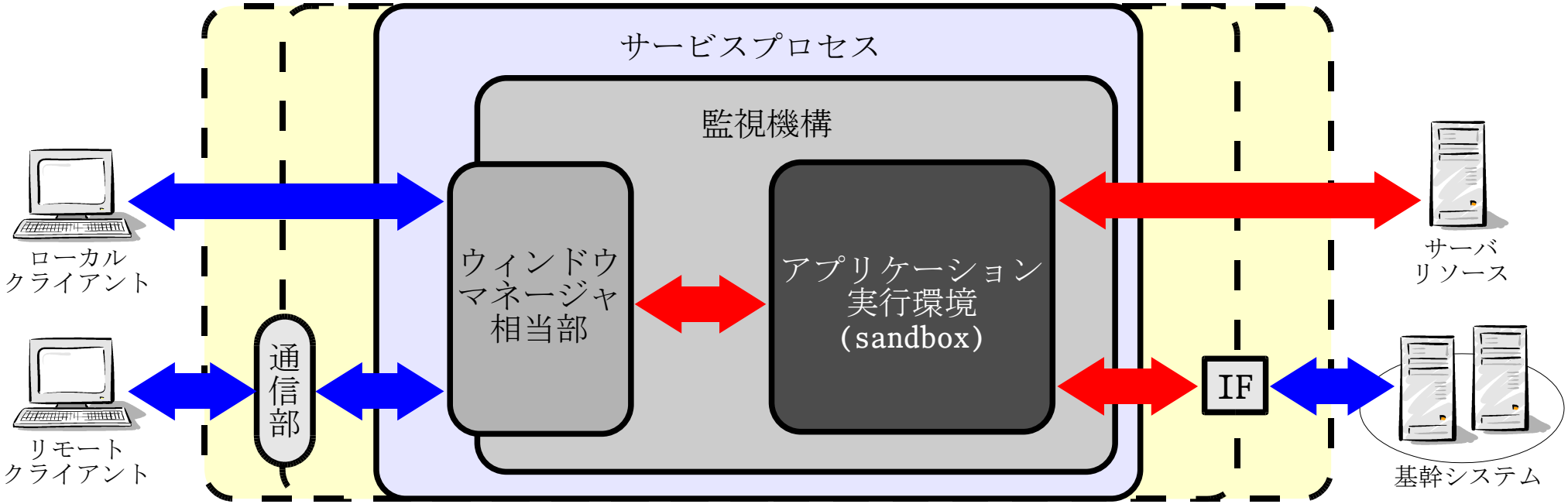
考えられるアプリケーション例

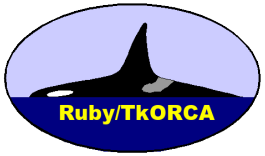
- 大学の研究室等での GUI を用いたデモの外部公開
  - 特殊なライブラリを駆使することも可能
  - 秘密にしておきたい情報は隠したままにできる
- GUI の表現力と操作性とを持った業務系 / 情報系システム
  - クライアントへの低い性能要求
    → 多種のプラットフォームや旧型機の有効活用
  - アプリケーションの導入や更新・改訂が低コスト
  - クライアントには情報が蓄積されない
    → クライアント PC 盗難による情報漏洩の回避
  - フレームワークが規定するのはインターフェースのみ
    → 背景となる基幹システムの選択は自由
- 個人作成の GUI ツールのリモート操作アプリケーション化
  - 作り直しの心配もなく，非常に手軽に実現可能



実現のための技術的な要請

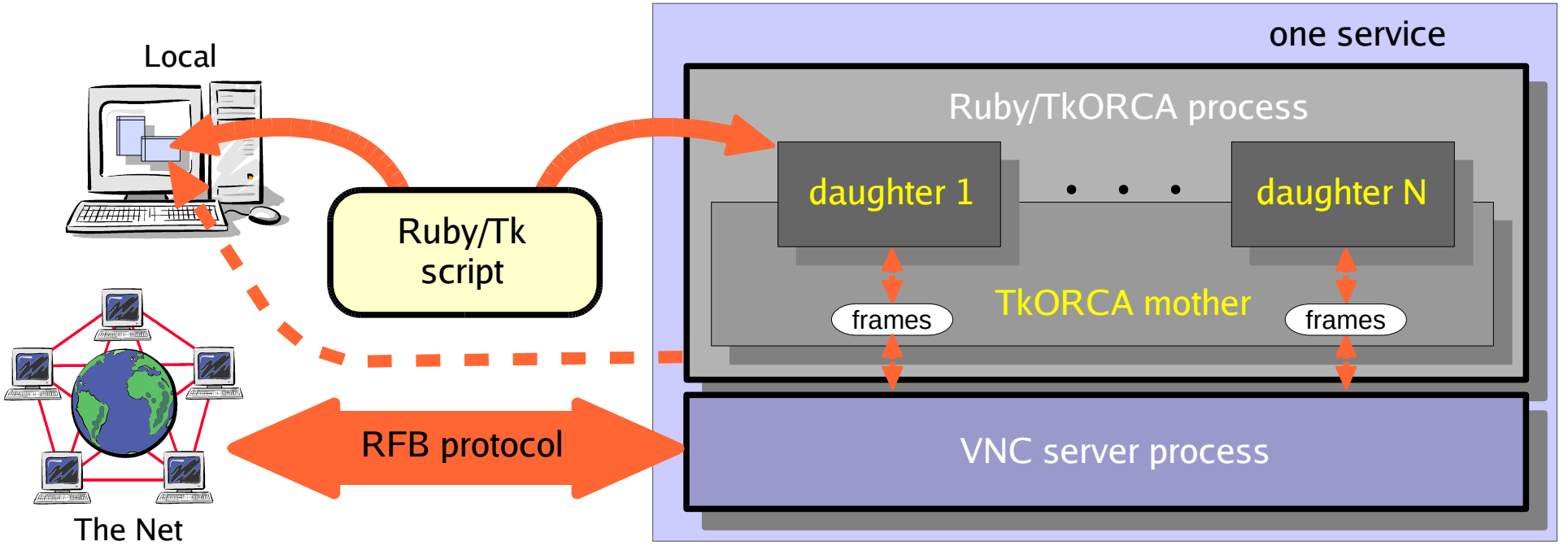
- ローカルのウィンドウシステムで動く GUI アプリをそのままに稼働させることができる実行環境
- 自動的なセキュリティ配慮と監視の実現
- 実行環境と同じく監視下で働くウィンドウマネージャ相当部
- 負荷軽減のために１プロセスとしつつも，各部の明確な分離の実現

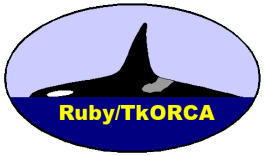




どうやって実現するのか？

- Ruby/Tk によって次のものを一つのプロセス上に実現
  - mother ：ウィンドウマネージャ機能とプロセスの総括
  - daugher ：公開用アプリケーション実行環境 (複数可)
- 基本はこのプロセスと VNC サーバプロセス （ RFB 通信専用であり，ウィンドウマネージャは非稼働） との二つで 1 サービス
- 公開する Ruby/Tk スクリプトを daughter に読み込んで実行



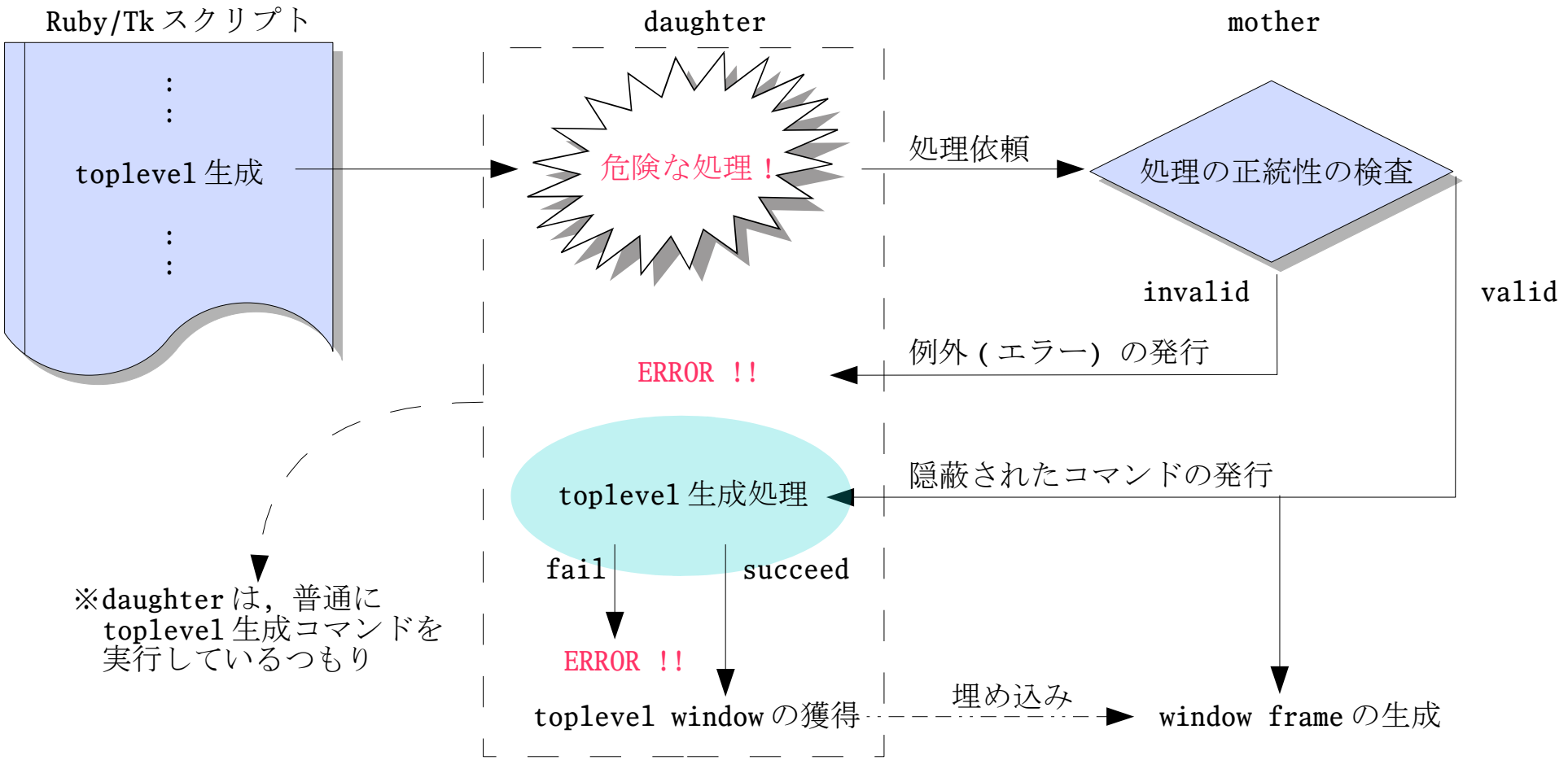


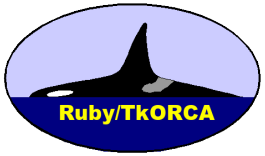
実行監視例：トップレベルウィンドウの生成

トップレベルウィンドウの生成を無条件に認めるのは危険

↓

生成は監視の下で

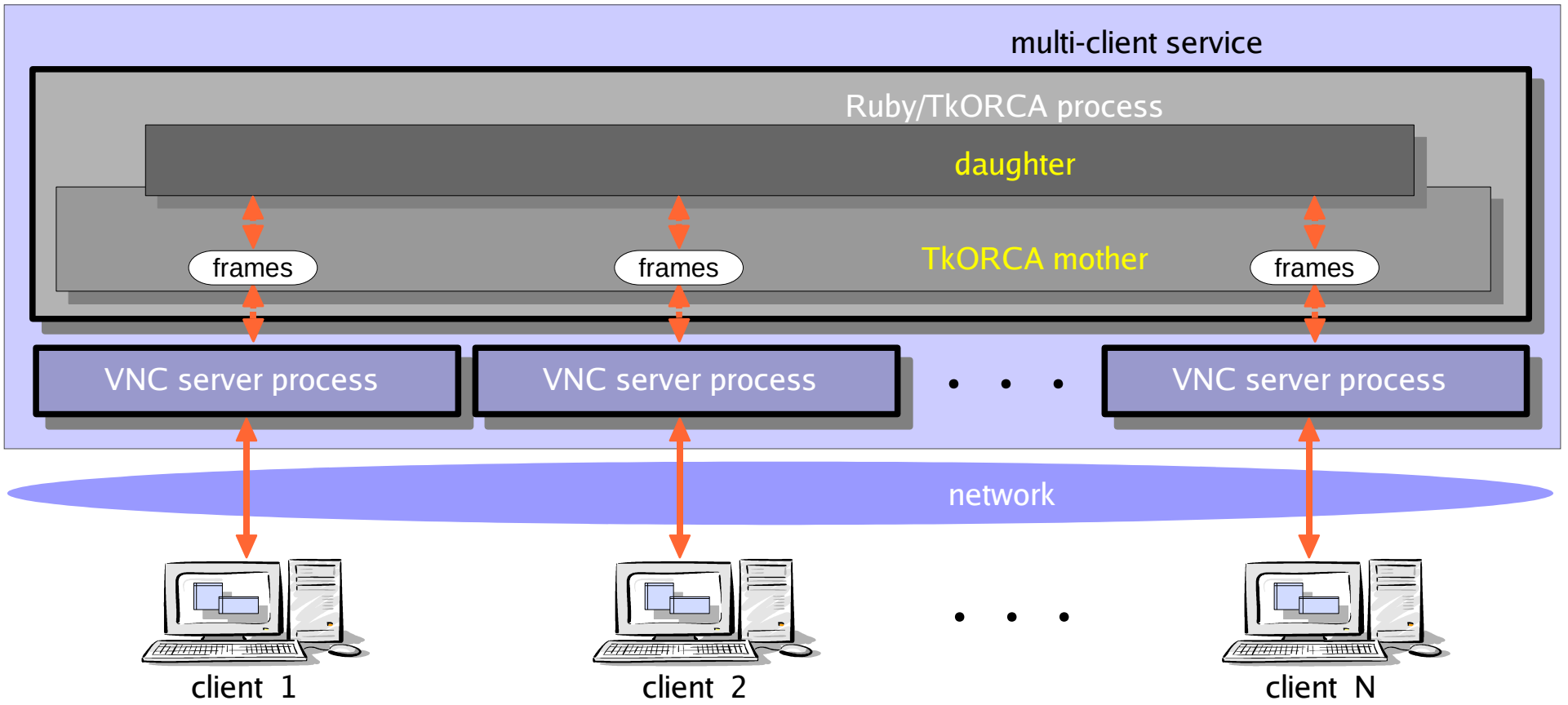


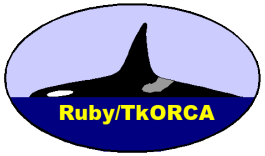


運用形態のバリエーションとその利点(1)

一組の mother/daughter で複数のクライアントにサービス

- daughter 側から見た場合には，単に複数のウィンドウを開いているのみ
- 各クライアント上での操作は event queue 上でシリアライズされるため，複数クライアントを連携させる処理の記述が簡単になる

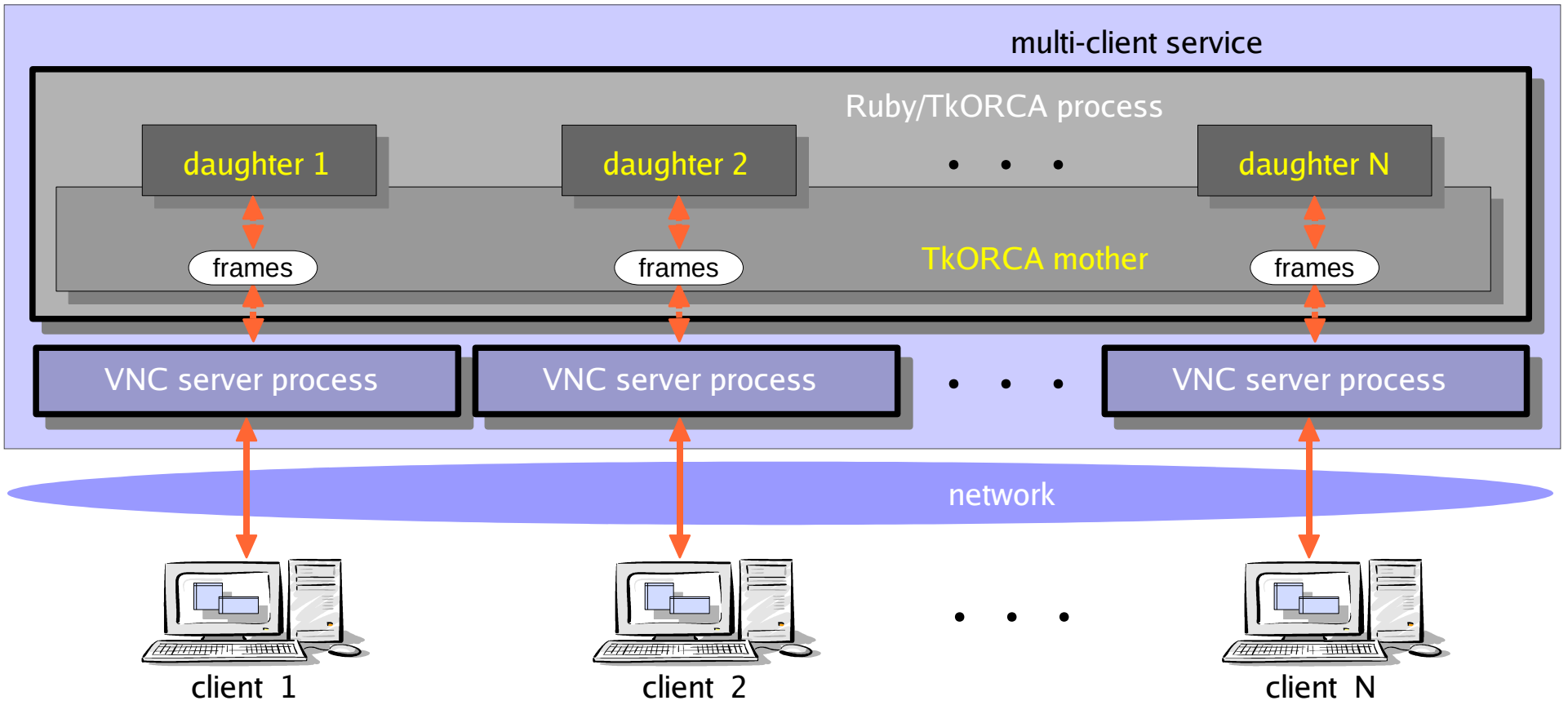


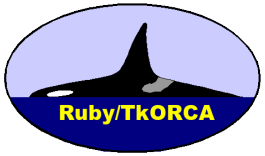


運用形態のバリエーションとその利点 (2)

一つの mother で複数の daughter/client の組にサービス

- 各 daughter は個々に独立した (干渉できない) 状態
- mother を介しての疑似プロセス間通信の他, mother の監視・干渉による daughter 間連携が可能

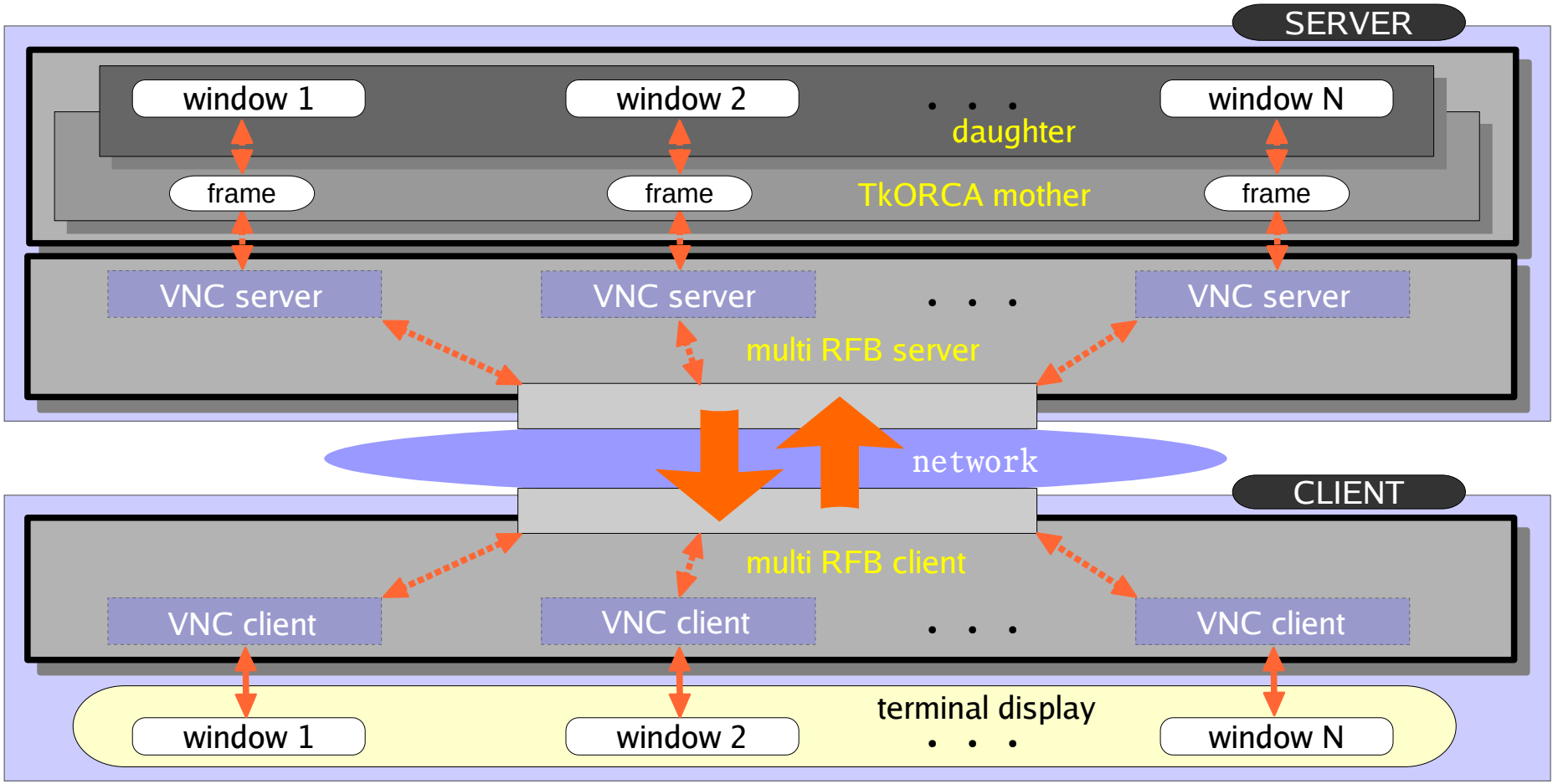


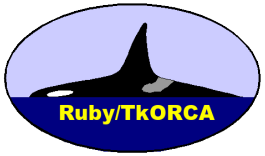


運用形態のバリエーションとその利点 (3)

マルチウィンドウ RFB サービス

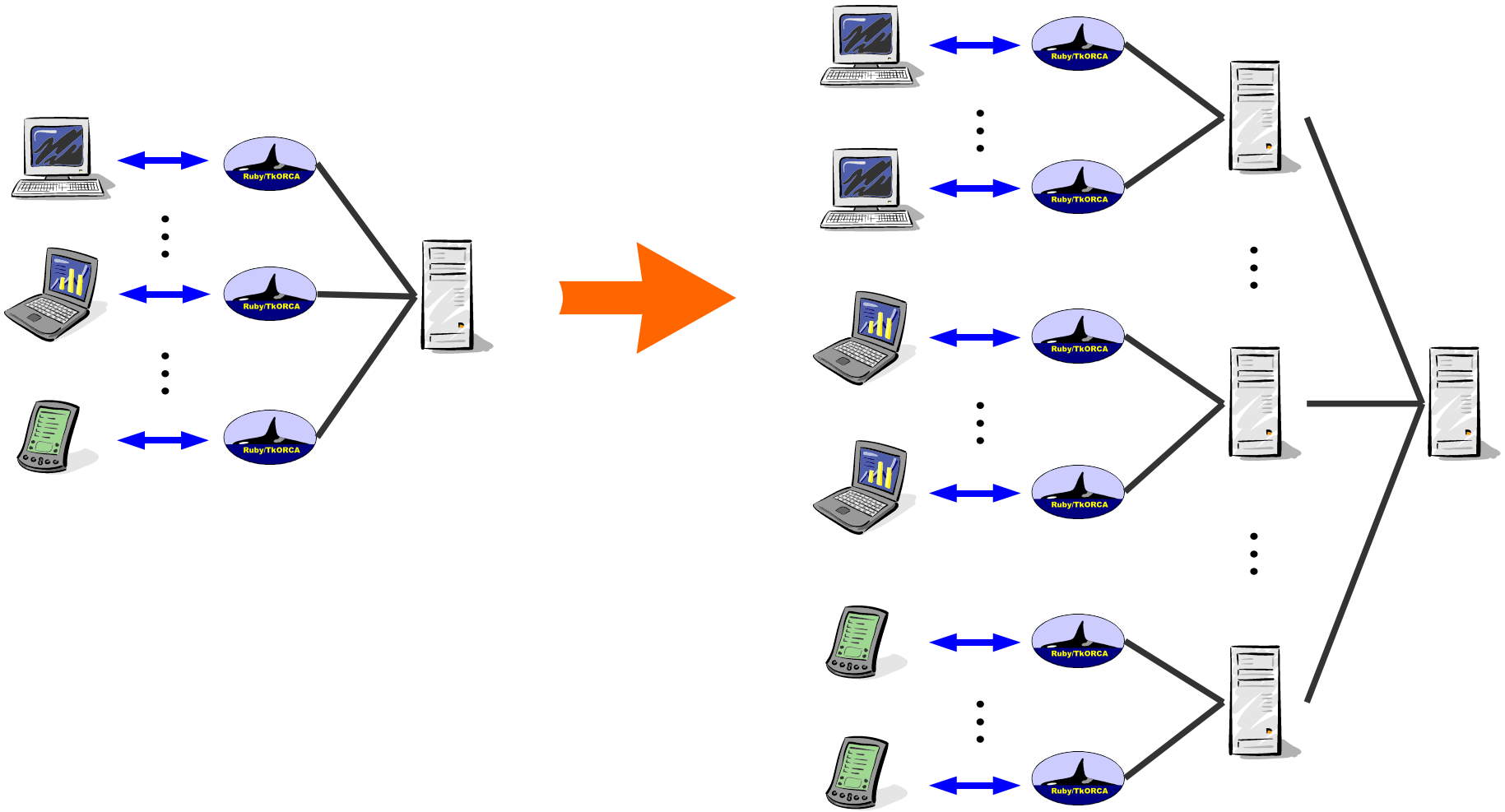
- １組の mother/daughter による複数クライアントサービスの応用
- １ウインドウ = １クライアント相当として RFB プロトコル通信を束ねて送り，クライアント側で別々のウィンドウに展開
- ウィンドウ移動やアイコン化等はクライアント側に処理を依頼

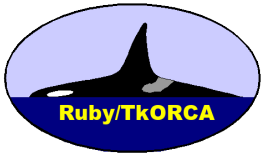




サービスのスケールアップ

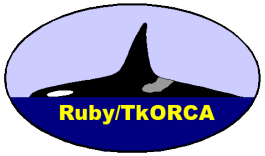
- 本フレームワークはアプリケーションのフロントエンド部であるので，スケールアップは主にバックエンドの強化で行う





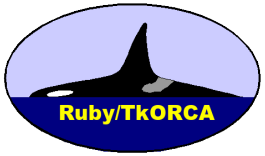
なぜ「Ruby/Tk」なのか？

『Ruby/Tk だからこそ実装可能』
と言えるような利点を持つため



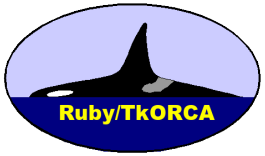
なぜ「Ruby/Tk」なのか？

- mother & daughter の分離を実装できる機能の存在
  - MultiTkIp クラスによる複数の Tk インタープリタ (IP) の駆使
    - IP は enclose された ThreadGroup で分離
    - Ruby/Tk スクリプトを IP の文脈で評価可能
  - ウィジェット管理が IP ごとに分離
    - ウィジェットオブジェクトは IP の情報を持たない
      → 他の IP のウィジェットオブジェクトを入手しても操作不能
  - safe slave IP を生成可能 (Ruby と Tk との両方でのセーフ機構)
  - master IP から slave IP の監視・操作が可能
    - 操作可能な IP オブジェクトは直接の子のスレーブ IP のみ
      → 親や姉妹の IP オブジェクトを入手しても操作不能



なぜ「Ruby/Tk」なのか？

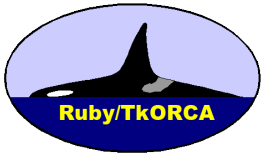
- ウィンドウマネージャの代わりを務めることができる機能の存在
  - frame ウィジェットのコンテナ機能
  - canvas ウィジェットの埋め込みウィンドウ機能
  - canvas のスクロールによる表示可能サイズよりも広い画面領域
- Tk への埋め込みが可能な非 Tk ライブラリも多い
  - 埋め込みさえできれば，非 Tk の画面出力でも支配下に置ける
  - safe Tk での稼働を想定していないものも多い点には要注意
    - → Ruby のセーフ機構によってカバーする必要があるケースも



『Tk』では能力不足ではないのか？

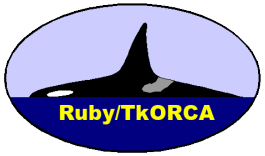
- 『Tk は処理速度が遅いでしょ』
  - 近年の新しい GUI ツールキットに比べて遅いのは確か
  - 通常の律速段階は RFB プロトコルによる通信なので，ほとんど問題なし
- 『Tk の見掛けって古めかしいよね』
  - Tcl/Tk 8.4 までの標準のデザインについてはその通り
  - スタイリングエンジンである Tile 拡張が利用可能（Ruby/Tk でも利用可能)
  - Tcl/Tk 8.5 では改善への動き (Tile 拡張の標準添付化も決定)
- 『単純なウィジェットしかないし，3D 表示もダメでは? 』
  - 欲しいものは組み合わせて作るというのが Tk の考え方
  - 面倒なら，多様な Tk 拡張や Tk と連携できるライブラリを使えば良い
  - 3D GUI の意味ならその通りだが，その点では他も大差なし

→ いずれもさして大きな問題ではない



対応可能な公開アプリケーションの条件

- GUI の最表層が Ruby/Tk で作られていること
  - 表示する画面の大きさに制限なし
    (クライアントではスクロール表示となる)
  - 各種 Tk 拡張ライブラリの使用可能
  - 描画を Tk のコンテナに埋め込めるライブラリも使用可能
- 描画を行わないライブラリには利用制限なし
  - 実行状況の監視には制約あり
- サーバ・クライアント間のデータ転送は不可
  - 並行してコネクションを生成するならその限りではない
- 動画やサウンド等を含むリアルタイム処理は不可
  - 簡単なアニメーションの表示程度は可能



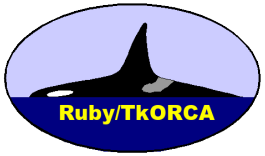
対応可能な公開アプリケーションの条件

ただし…

本フレームワークのみでアプリ全体を組む必要なし

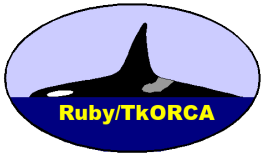
- 使っている RFB プロトコルはオープンなプロトコル
- クライアント側はコンパクトで軽負荷
- VNC ビューア相当をアプリケーションに組み込んでも負担は小さい
- 既存の Java アプレット版 VNC ビューアの利用や同等品作成により，Web アプリの一部に埋め込むことも可能

→ アプリ上の必要な部分のみでの柔軟な活用が可能



実際に実現できるのか？ 試作品はあるか？

- ローカルでのサンプル実行の実例



実際に実現できるのか？ 試作品はあるか？

- ローカルでのサンプル実行の実例
- 遅いマシンだが，コンセプト実証用実験サーバが存在
  - Web ブラウザによるアクセス
    (Java アプレットが実行可能であること)

    http://131.206.154.81/

  - VNC ビューアによるアクセス

    IP addr:131.206.154.81　Port:5933

※ いずれの場合も IP addr:131.206.154.81 , Port:5933 へのパケットをファイアウォール等が通過させることが必要

ご清聴，ありがとうございました