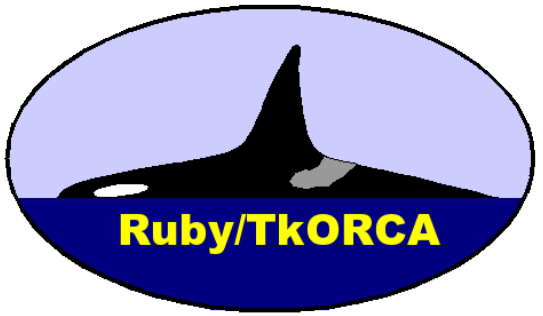
[2005年度下期未踏ソフトウェア創造事業]

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の開発

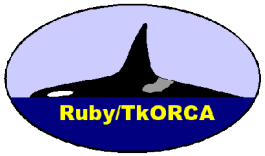
～ Ruby/TkORCA (Ruby/Tk On RFB Canvas) ～



永井 秀利

九州工業大学 情報工学部 知能情報工学科

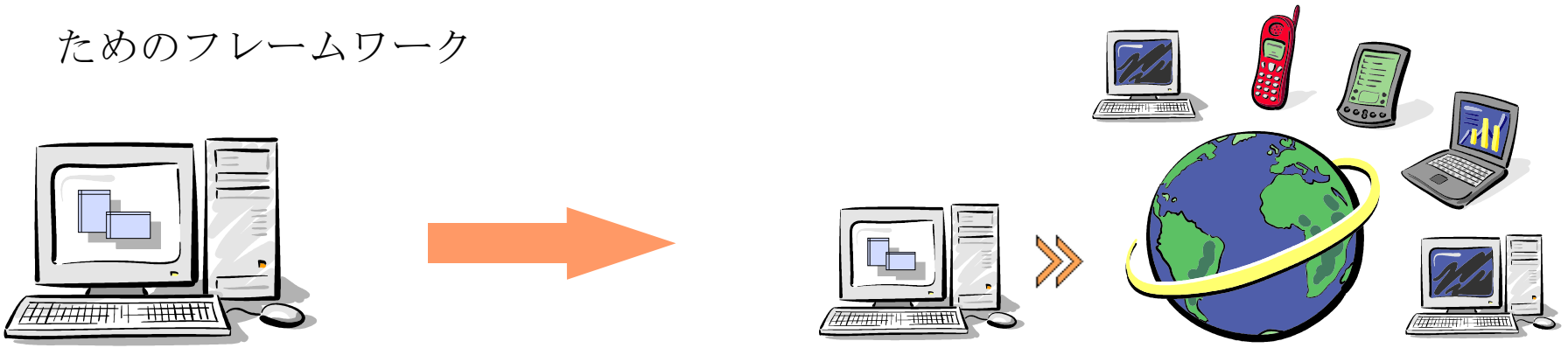
2005/12/26

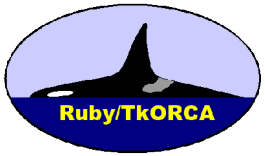


開発を目指すフレームワークとは？

- 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

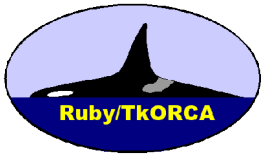




一言で言うなら．．．

どこでもGUI!!

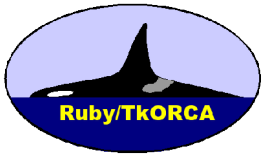
（ GUI,Anyware!! ）



どんな人にとって有益か？

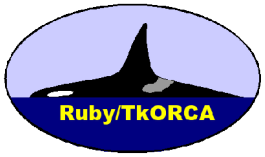
『GUI + ネットワーク公開』

というキーワードに興味を持つすべての人に



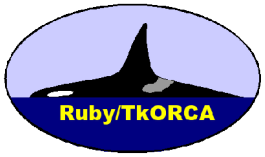
現在のネットワークアプリにおける問題点

- 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
  (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 GUI ネット利用の枠組における問題点

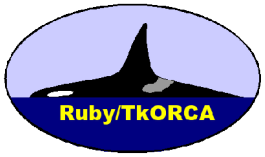
- ローカルの個々の GUI アプリケーションのみを広く安全に公開するための枠組の不在
- X プロトコル
  - クライアントに対する制約が大きい上，プロトコル自体が重すぎる
- Windows リモートデスクトップ
  - マシンの遠隔操作用であり，特定アプリのみのネット公開目的には役に立たない
- VNC (RFB プロトコル)
  - 一般的な利用形態はマシンの遠隔操作 → 他の利用形態への想定や配慮が希薄
  - 運用上，ウィンドウマネージャが癌
    - かなり大きな操作権限を与えてしまうにもかかわらず，動作の監視ができない
    - 稼働プロセス数が増えるなど，結構重い上に起動も遅くなる
    - 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）



本フレームワークによればどう変わるのか？

- 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

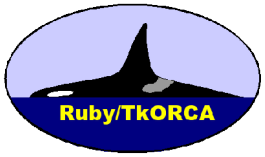
→ 開発の速度や効率の向上



本フレームワークによればどう変わるのか？

- 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の監視下の 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

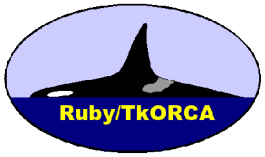
→ セキュリティ確保を支援する機能により
安全性向上に要するコストを低減



本フレームワークによればどう変わるのか？

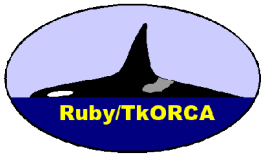
- 容易な公開設定と柔軟なクライアント対応
  - 公開時環境と同等のテスト環境で確認済みのため，公開や更新の作業はサーバでの登録を書き換える程度
  - thin client であるため，パソコンに限らずPDA クラスの機器でもクライアントになることが可能
  - クライアントへの依存性がほとんどないため，サーバ上のアプリケーションを更新してもクライアントの更新は必要なし

→ ニーズに対しての極めて高い即応性



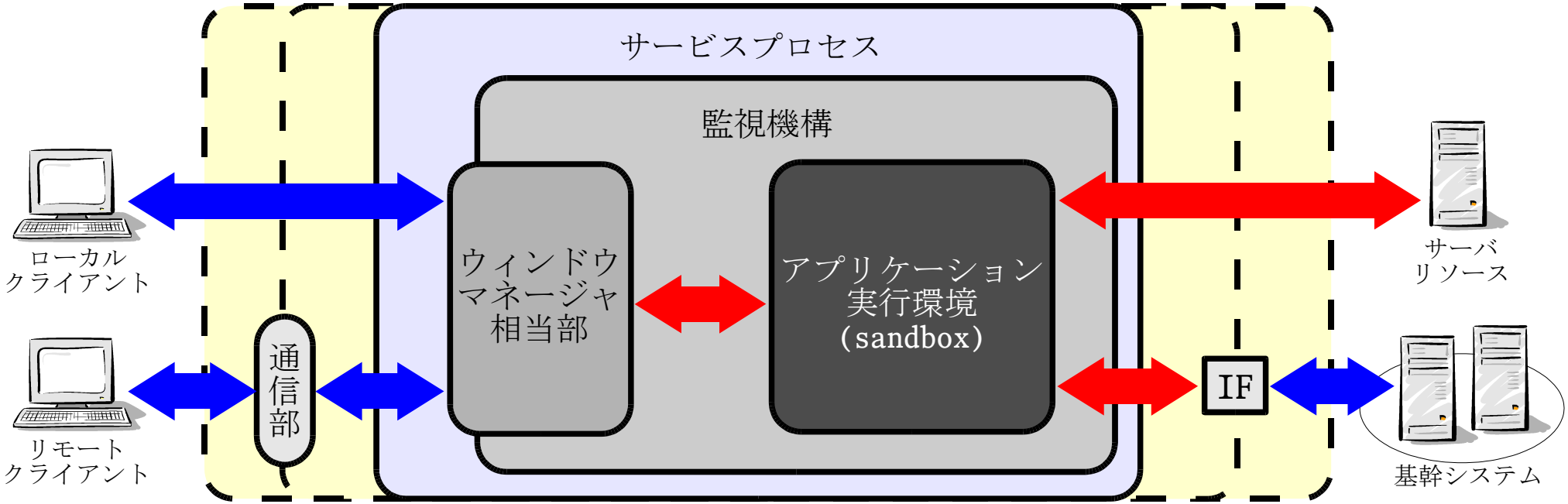
考えられるアプリケーション例

- 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



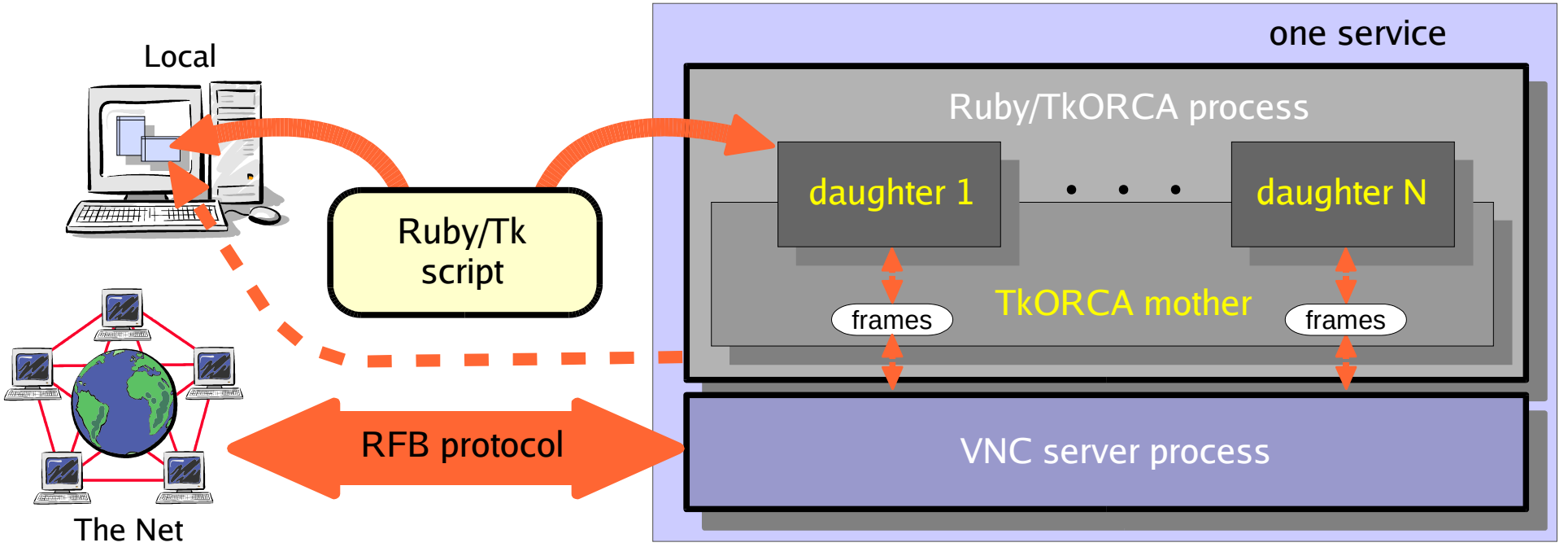
実現のための技術的な要請

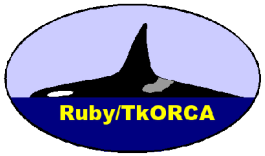
- ローカルのウィンドウシステムで動く GUI アプリをそのままに稼働させることができる実行環境
- 自動的なセキュリティ配慮と監視の実現
- 実行環境と同じく監視下で働くウィンドウマネージャ相当部
- 負荷軽減のために１プロセスとしつつも，各部の明確な分離の実現



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

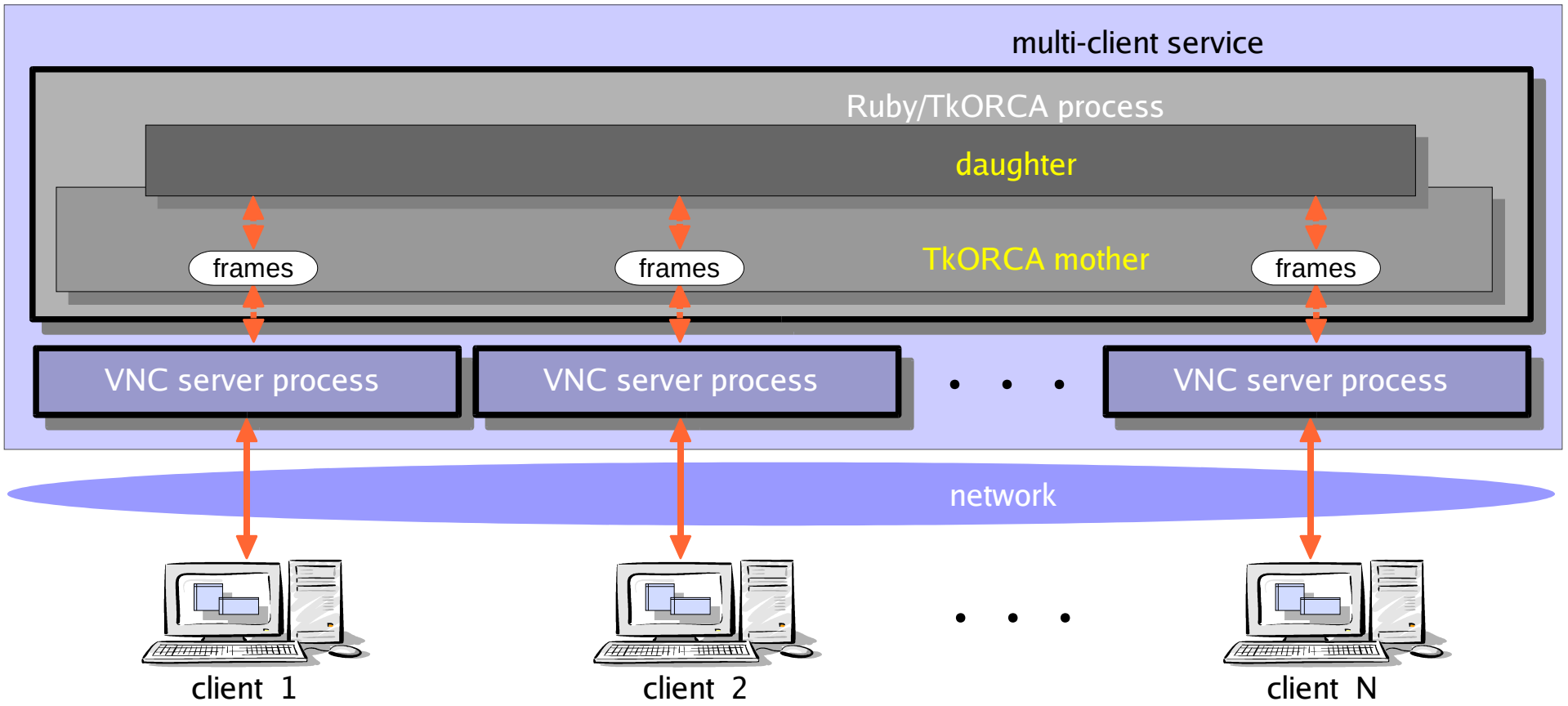


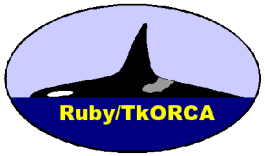


運用形態のバリエーションとその利点 (1)

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

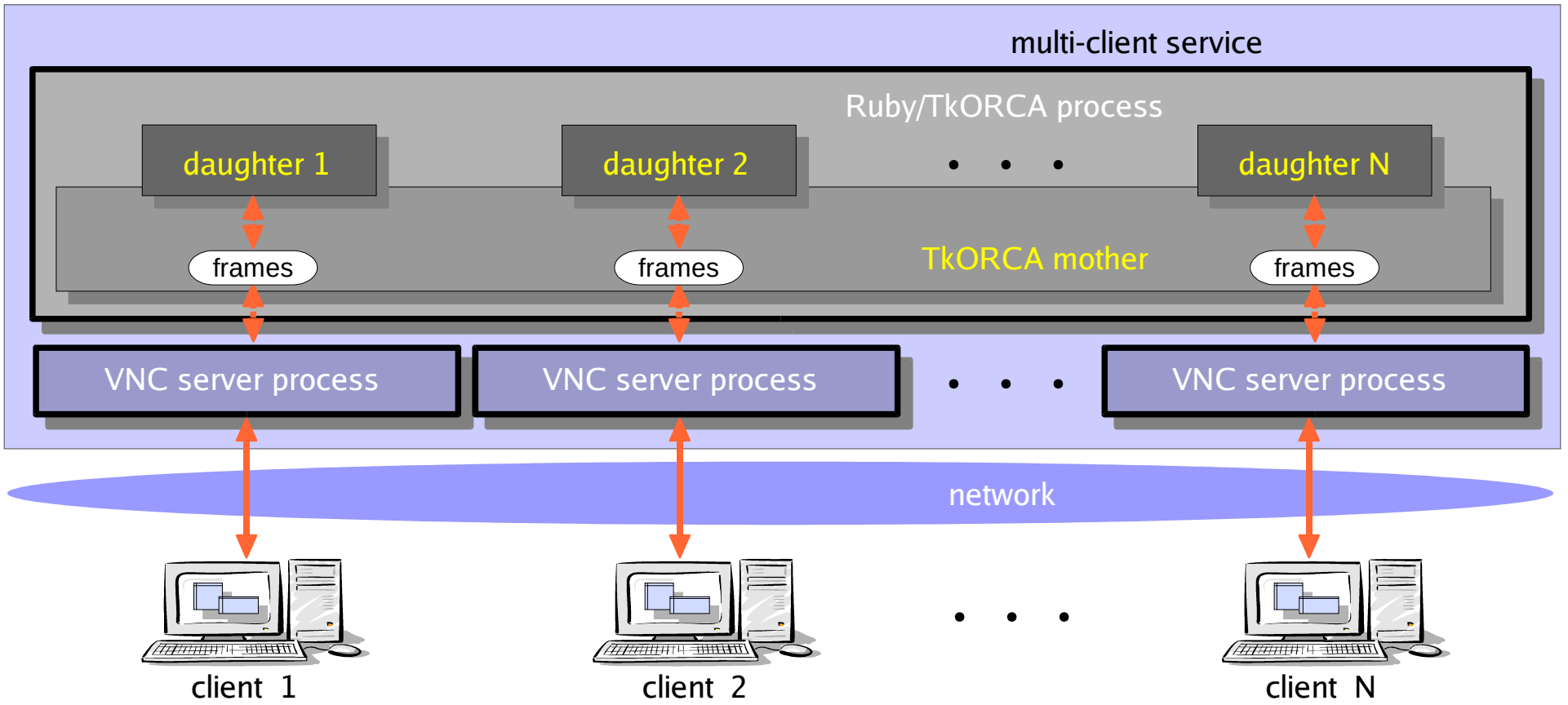




運用形態のバリエーションとその利点 (2)

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

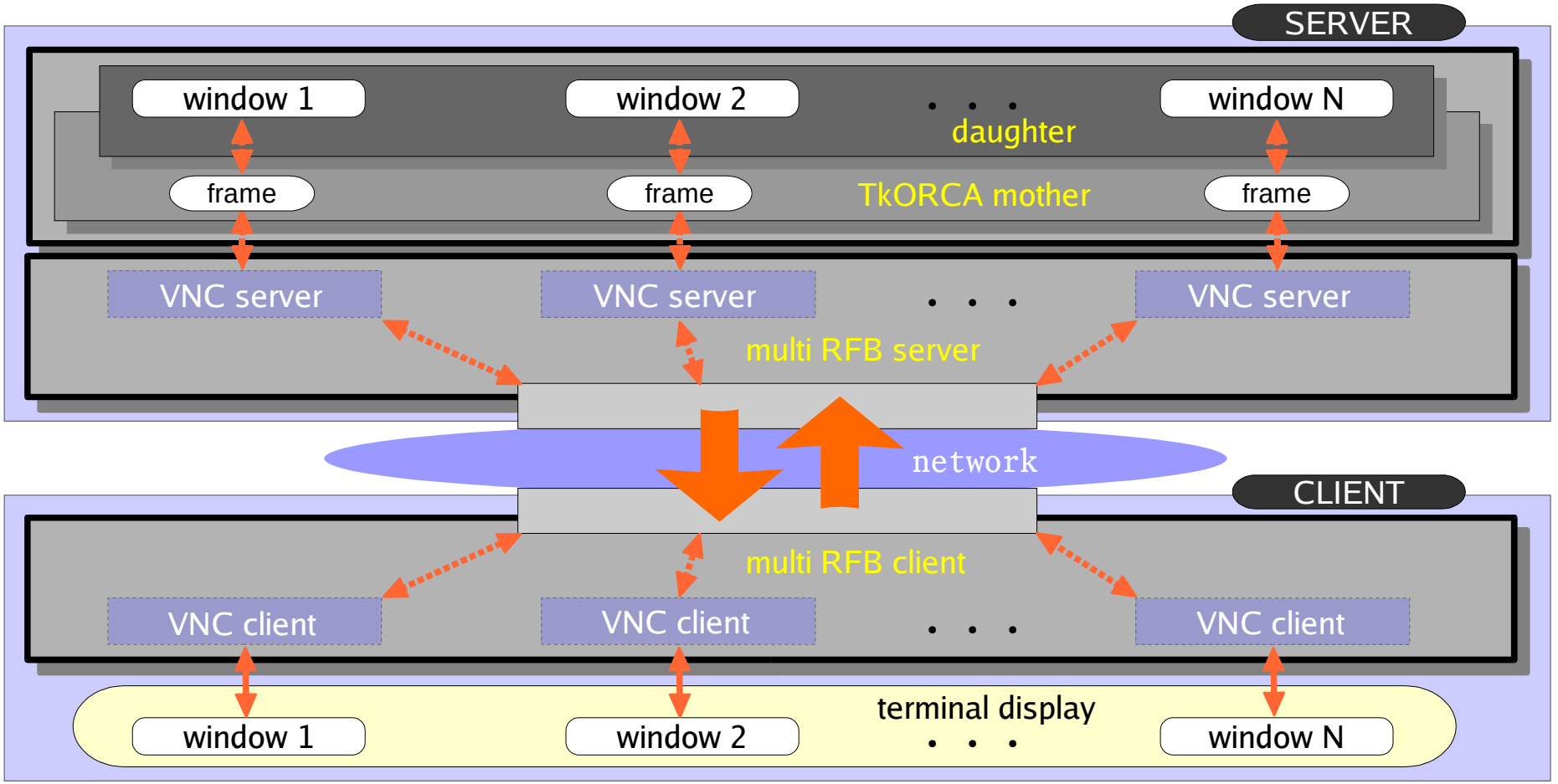
- 各 daughter は個々に独立した (干渉できない) 状態
- mother を介しての疑似プロセス間通信の他, mother の監視・干渉による daughter 間連携が可能

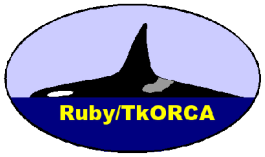


運用形態のバリエーションとその利点 (3)

マルチウィンドウ RFB サービス

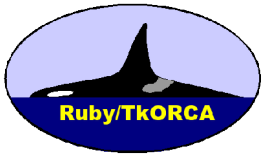
- 1 組の mother/daughter による複数クライアントサービスの応用
- 1 ウインドウ = 1 クライアント相当として RFB プロトコル通信を束ねて送り，クライアント側で別々のウィンドウに展開
- ウィンドウ移動やアイコン化等はクライアント側に処理を依頼





なぜ「Ruby/Tk」なのか？

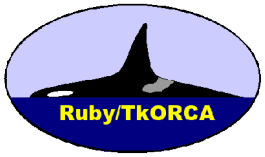
『Ruby/Tk だからこそ実装可能』
と言えるような利点を持つため



Ruby/Tk を用いることの利点

- mother & daughter を実装できる機能の存在
  - safe Tk と Ruby のセーフレベルとによる二重の安全機構
  - Tk インタープリタごとに分離されたウィジェット操作権限
  - 監視や操作の可否を決める Tk インタープリタ間の支配関係
  - 権限のないオブジェクト（Tk インタープリタやウィジェット）は，たとえ入手できても操作不能という安全性

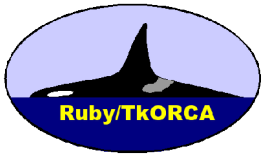
- その他の利点
  - ウィンドウマネージャの役割を務めうる機能の存在
  - 様々なプラットフォームで動作するポータビリティの高さ
  - Tcl/Tk の歴史の長さにより，連携可能なライブラリも多い



『Tk』では能力不足ではないのか？

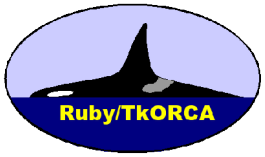
- 『Tk は処理速度が遅いでしょ』
  - 近年の新しい GUI ツールキットに比べて遅いのは確か
  - 通常の律速段階は RFB プロトコルによる通信なので，ほとんど問題なし
- 『Tk の見掛けって古めかしいよね』
  - Tcl/Tk 8.4 までの標準のデザインについてはその通り
  - スタイリングエンジンである Tile 拡張が利用可能（Ruby/Tk でも利用可能)
  - Tcl/Tk 8.5 では改善への動き (Tile 拡張の標準添付化も決定)
- 『単純なウィジェットしかないし，3D 表示もダメでは? 』
  - 欲しいものは組み合わせて作るというのが Tk の考え方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



対応する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



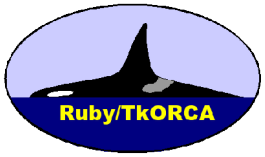
対応する公開アプリケーションの条件

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

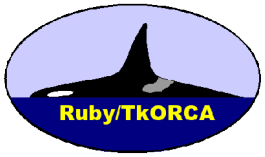
- 使っている RFB プロトコルはオープンなプロトコル
- クライアント側はコンパクトで軽負荷
- VNC ビューア相当をアプリケーションに組み込んでも負担は小さい
- 既存の Java アプレット版 VNC ビューアの利用や同等品作成により，Web アプリの一部に埋め込むことも可能

→ 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？

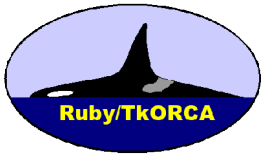
- ローカルでのサンプル実行の実例
- 遅いマシンだが，コンセプト実証用実験サーバが存在
  - Web ブラウザによるアクセス
    (Java アプレットが実行可能であること)

    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

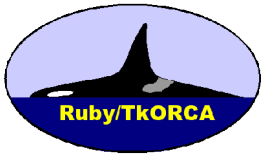
ご清聴，ありがとうございました



開発予定項目

- Ruby/TkORCA 本体
  - ウィンドウコントロールの実装 (試作での実装済みはごく一部)
  - アプリケーションからの要求への対応
  - 入出力コンソールの代用品 (安全のため shell の稼働は禁止)
  - 危険な命令のリストアップとそれらの wrap （必要なら Ruby/Tk の修正)
  - 監視や実行制約の機能 (API) の実装
  - 複数 daughter への対応
  - 見掛け (スキン) のカスタマイズ機能
  - ドキュメント整備

- 本フレームワーク用の VNC サーバ
  - 試作品ではデモを実行するサーバマシンの設定に特殊な細工を施して実現
  - パッチを提供する？ VNC サーバ自体を提供する？
  - 本フレームワーク用サーバ設定ツールは提供を予定

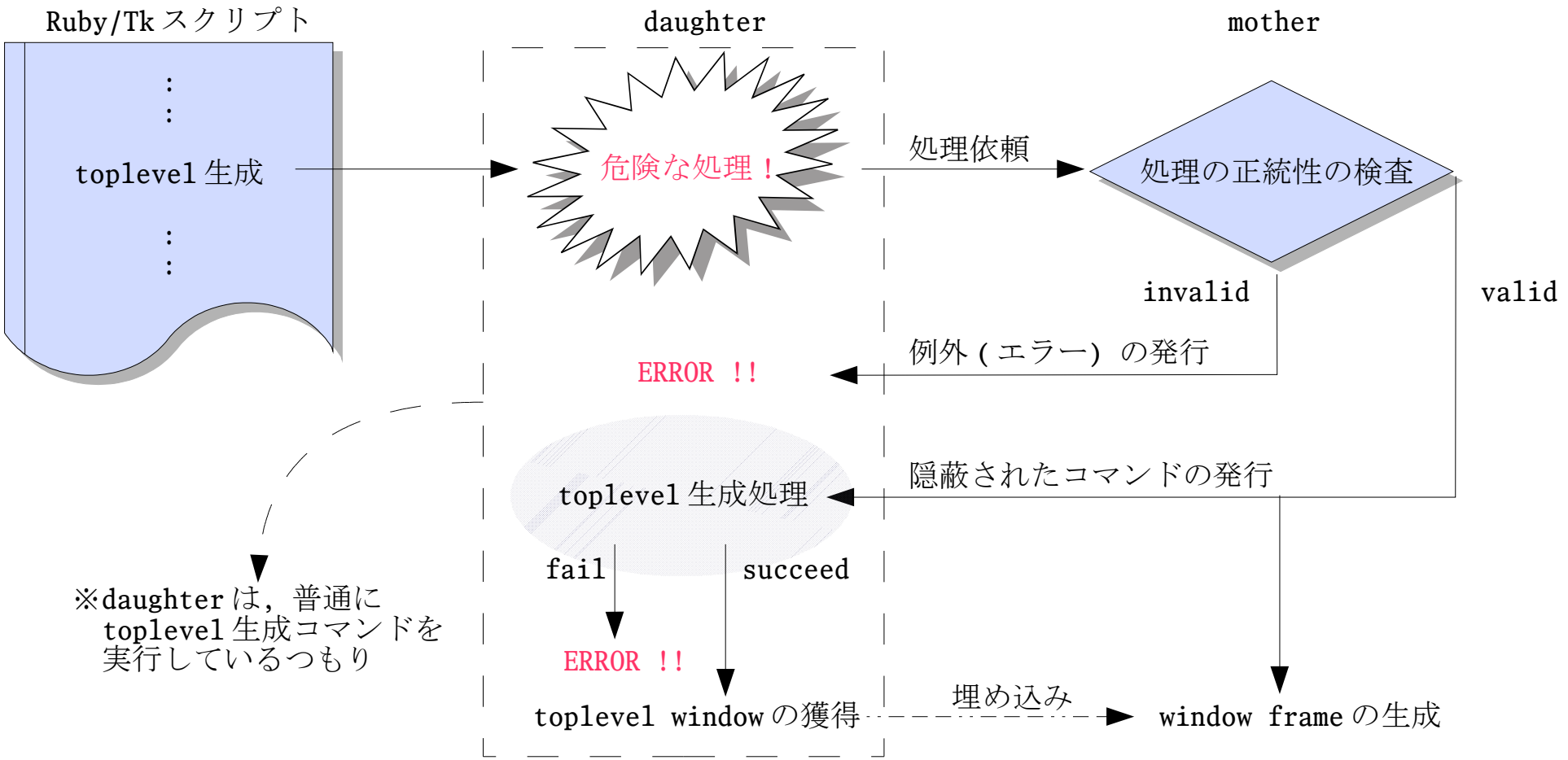


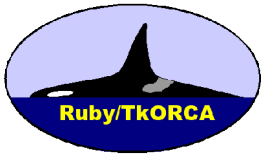
実装例：トップレベルウィンドウの生成

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

↓

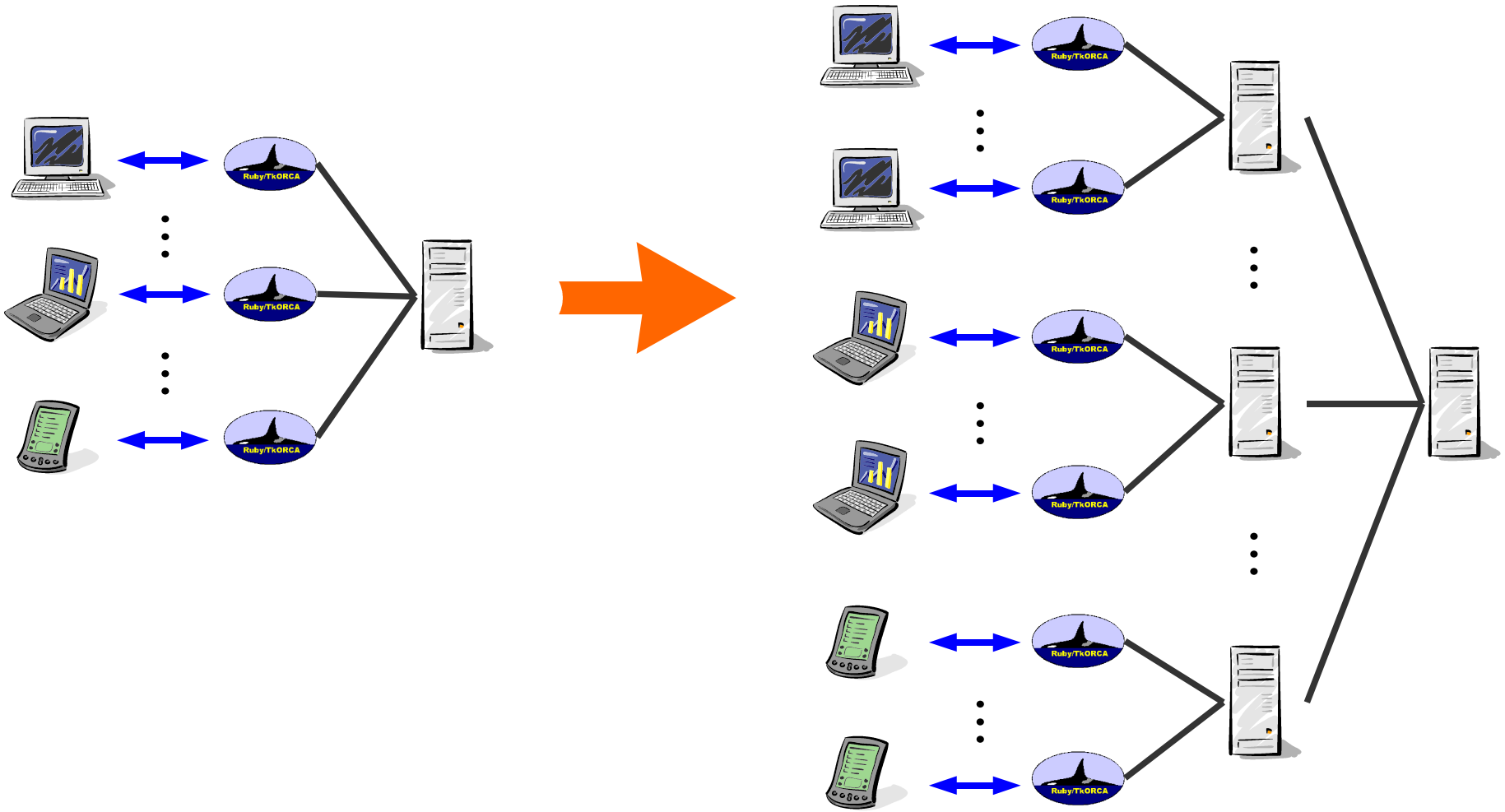
生成は監視の下で

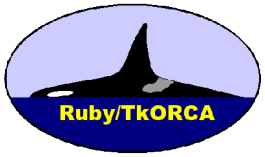




サービスのスケールアップ

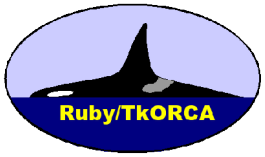
- 本フレームワークはアプリケーションのフロントエンド部であるので，スケールアップは主にバックエンドの強化で行う





なぜ「Ruby/Tk」なのか？

- mother & daughter 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
- 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



本フレームワーク利用時の基本的開発工程

(0) ローカルで稼働する GUI の作成

稼働しているものがすでに存在するなら作成の必要なし (そのまま利用)

(1) ローカルのウィンドウシステム上のテスト環境での稼働試験

(i) sandbox の制約に基づくセキュリティエラーのチェック

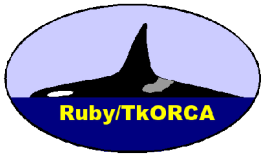
(ii) 監視オプションの設定 (必要なら)

(iii) 監視または制御スクリプトの作成 (特に必要なら)

(iv) アプリケーションソースの修正 (どうしても必要なら)

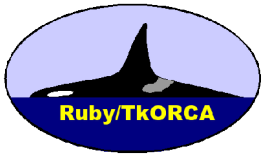
(2) サーバに設定して公開

テストは完了しているので，サーバ設定に追記する程度



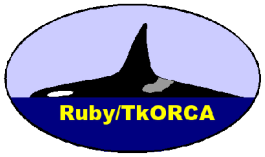
どのようなスキルが要求されるのか？

- 必須となるスキル
  - Ruby および Ruby/Tk の最低限の知識
  - サーバ設定ツールによるアプリケーション公開方法
  - サーバ設定に関連した最低限のネットワーク知識（サーバ管理者のみ）
- 推奨されるスキル（場合によっては必須）
  - safe Tk および Ruby のセーフレベルの制約や決まりごとの理解
  - 本フレームワークにおける監視や制御の設定項目や API の理解
  - 監視や制御の内容によっては Ruby/Tk での Tk インタープリタ操作方法
- 必ずしも必要ではないスキル
  - 本フレームワーク専用の GUI 作成技術
  - VNC などの基盤となっている部分の知識



将来は？

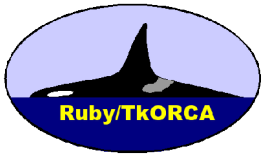
- サーバ・クライアント間データ転送機能の提供
  - クライアント側アプリの強化は必要となるが，クライアントへの性能要求を強めるほどではないはず
- Tk の描画処理部分と RFB サービスとの一体化
  - GUI 部が pure Ruby/Tk なら， 1 プロセスだけでサービス提供可能
  - サーバ負荷の軽減に直結
- セッションの継続 (GUI の永続化)
  - 継続可能性は条件付きとなるが，現在の GUI の構造を調査して保存，後にそれに基づいて再構築することで永続化を図る



どんな人にとって有益か？

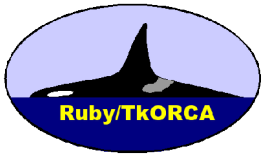
『GUI +ネットワーク公開』というキーワードに興味を持つすべての人に

- GUI を持ったデモやサービスを，ネットワークを介して広く公開したい人
- マルチウィンドウアプリを，ローカルからネット公開までシームレスに開発したい人
- 高度なインタラクティブ性を持つネットワークアプリケーション (Web アプリを含む) を簡単かつ気軽に製作し公開したい人
- それなりのインタラクティブ性は持たせたいが，Ajax のようなものは面倒だという人
- クライアントに情報提示はしたいが，データをクライアントに直接渡したくはない人
- 自作の GUI 付きツールを，ローカルでも外からでも同じに使えるようにしたい人



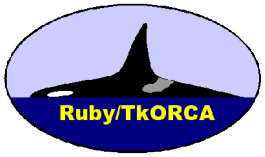
既存のアプローチでは？

- Ajax 等の thick client アプローチ（個別に独自のプロトコルを使用する場合を含む）
  - 気をつかう部分が多く，開発が面倒！
  - ローカルで使うときには実行環境が不必要に重い！
  - クライアント部がオープンなのは，嬉しくないことも多い？
- X プロトコル
  - クライアント制約が大きい上，プロトコル自体が重すぎ！
- VNC（RFB プロトコル）
  - 有力候補だが，ウィンドウマネージャが癌！
    - 操作権限の与えすぎるにもかかわらず，動作監視ができない
    - 稼働プロセス数が増えるなど，結構重い
    - GUI 操作性維持には必須（Win では置き換えすらできない）



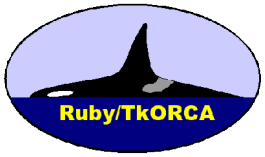
既存のアプローチでは何が問題か？

- Ajax 等の thick client アプローチ（独自プロトコル使用を含む）
  - 気をつかう部分が多く，開発や更新が面倒
  - 活用できるライブラリの制約大
  - ローカルで使うには，実行環境が不必要に重い
  - クライアントの能力に対する依存性や要求が大
  - クライアント部がオープンなのは，嬉しくないことも…
  - 与える情報の増大→クライアント上の情報蓄積量増大→情報漏洩の危険性の増大
- 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
- VNC（RFB プロトコル）
  - 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，結構重くなる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）



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
- 操作性向上を目指した場合の問題の増大
  - 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
  - クライアントへの性能要求の増大 (＝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データ量の増加による情報漏洩リスクの増大
- ローカルの個々の GUI アプリのみを安全に公開するための枠組の不在
- 画面出力を行う有用なライブラリの利用が困難



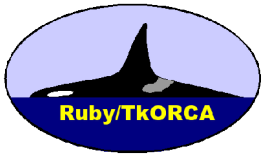
現在のネットワークアプリにおける問題点

- ローカルの GUI との間での操作性のギャップ
- 操作性を向上させる場合の問題の増加
- ローカルの個々の GUI アプリのみを安全に公開するための枠組の不在
- 画面出力を行う有用なライブラリの利用が困難

本プロジェクトで開発するフレームワーク

||

こうした問題に対する解の一つ

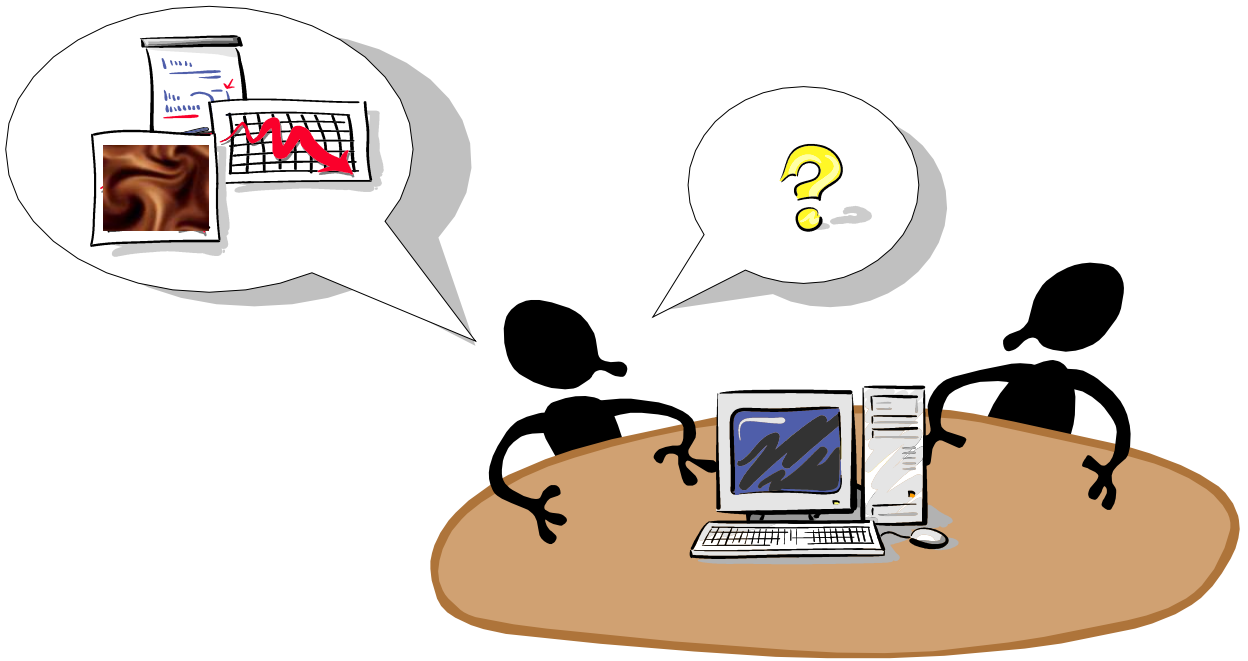


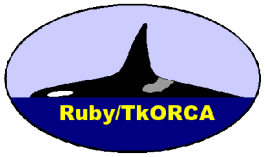
現在の状況では...

A:『こんなGUIで研究データ分析用ツールを作ってみたんだけど，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』

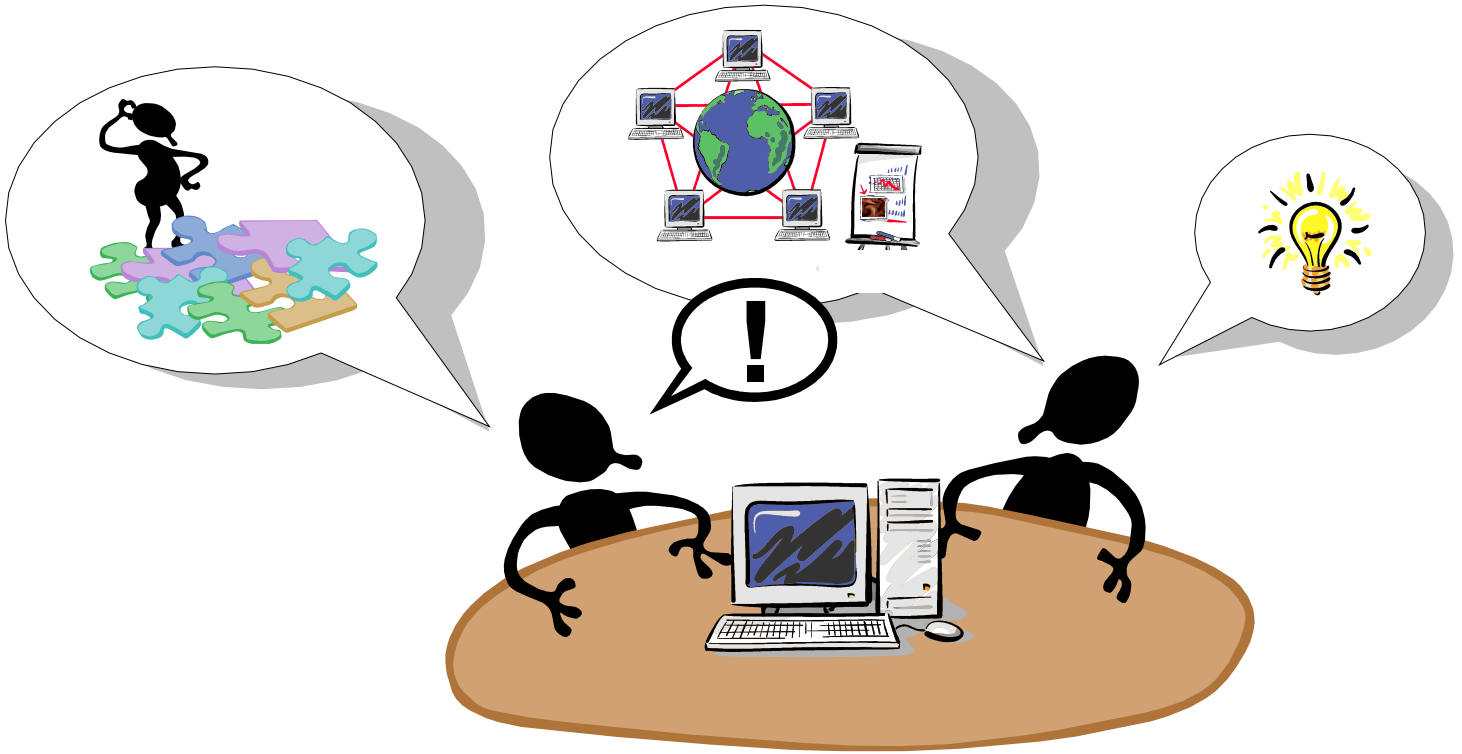


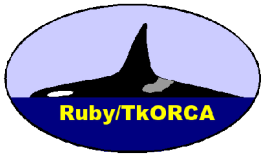


現在の状況では...

B: 『なるほど．あ，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？ 今はやりの 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．せっかく見付けたライブラリも使えなくなりそうだし...』



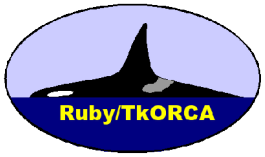


現在の状況では...

Ajax 等の thick client アプローチでの問題点

（個別に独自のプロトコルを使用する場合を含む）

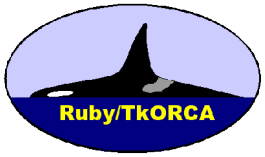
- 気をつかう部分が多く，開発や更新が面倒
- 活用できるライブラリの制約大
- ローカルで使うには，実行環境が不必要に重い
- クライアントの能力に対する依存性や要求が大
- クライアント部がオープンなのは，嬉しくないことも…



現在の状況では...

B: 『そうかぁ... なら, 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? 』

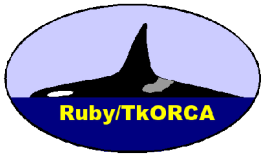


現在の状況では...

VNC で GUI を公開する際の問題点

→ ウィンドウマネージャが癌

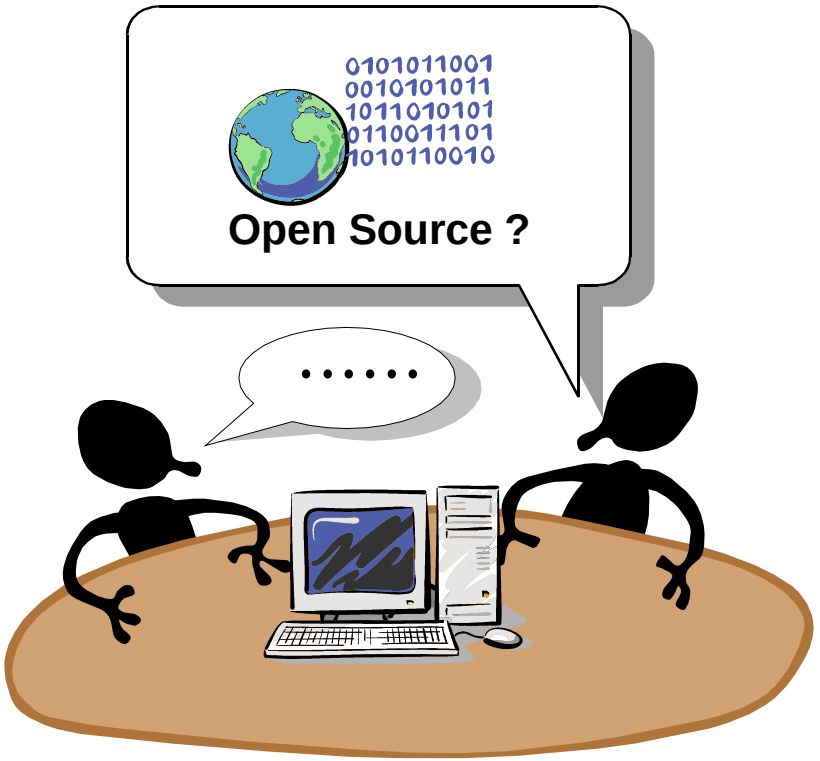
- かなり大きな操作権限を与えてしまうにもかかわらず，動作を監視することができない
- 稼働プロセスが増加するなど，結構重い上に起動も遅くなる
- とはいえ， GUI の操作性の維持のためには必須
  （Windows では置き換えすらできない）

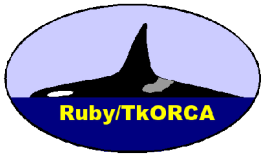


現在の状況では...

B: 『じゃあ，ソースを公開するしかないのかな』

A: 『でも研究上の重要な部分を含んでるしねぇ...
仮にソースを公開するとして，データは? なおのこと公開できないし，データなしじゃ意味ないよ? 』



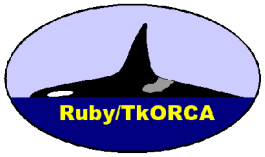


現在の状況では...

クライアントサイドアプリケーションの問題点

- ソースなりデータなりの形で情報をクライアントに与えざるを得ない
- 与えた情報はクライアントに蓄積される

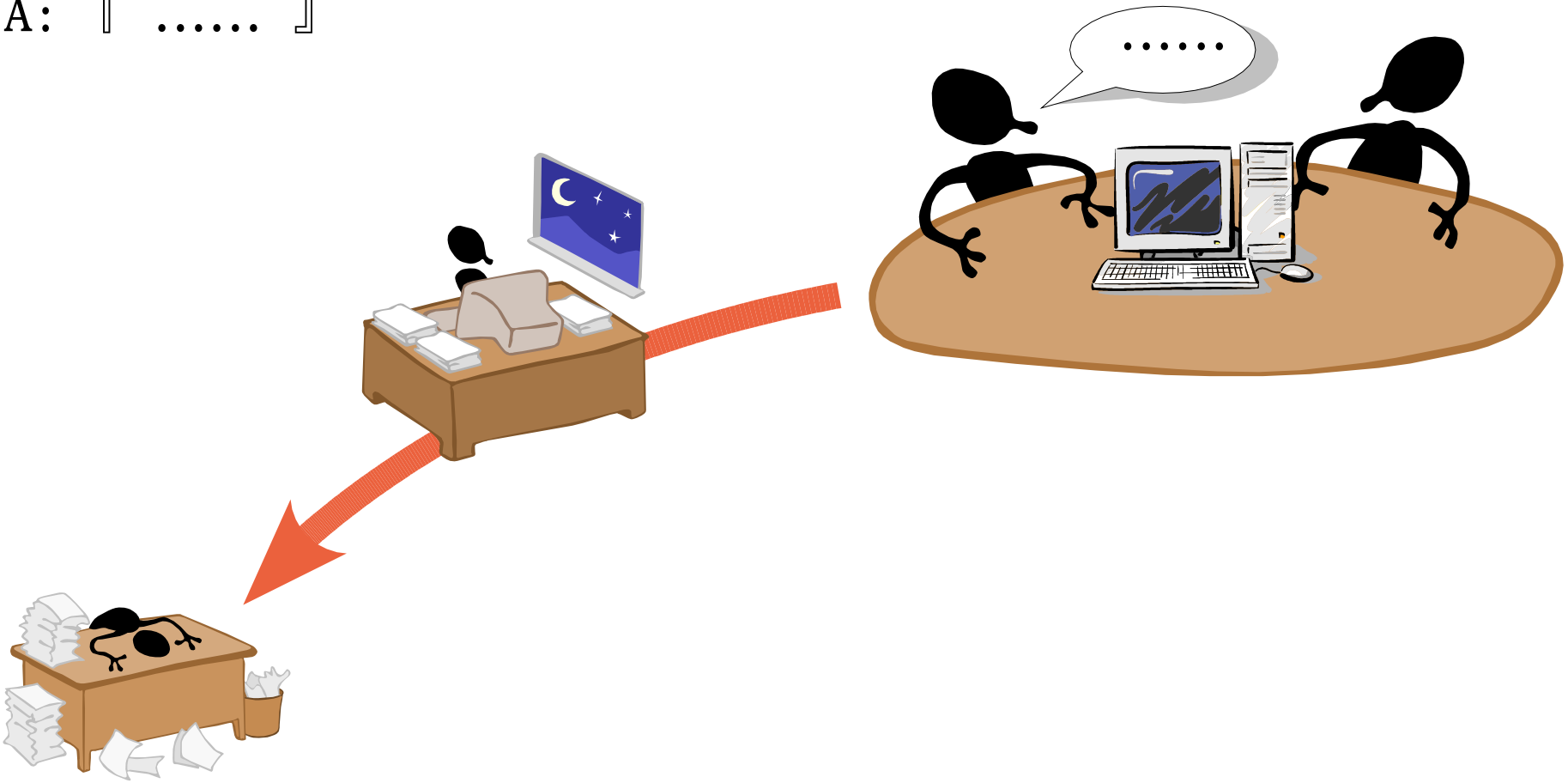
→ 情報漏洩の危険性



現在の状況では...

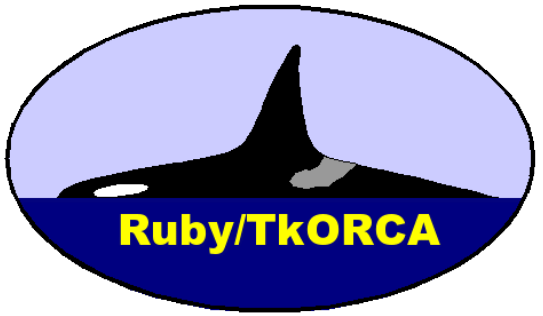
B: 『うーむ，どうにもならないか...
やっぱり公開用は別に作んなきゃいけないかなぁ...』

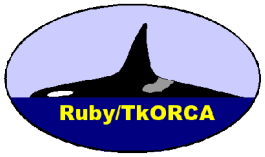
A: 『 ..... 』



このプロジェクトで開発する

フレームワークを使えば…



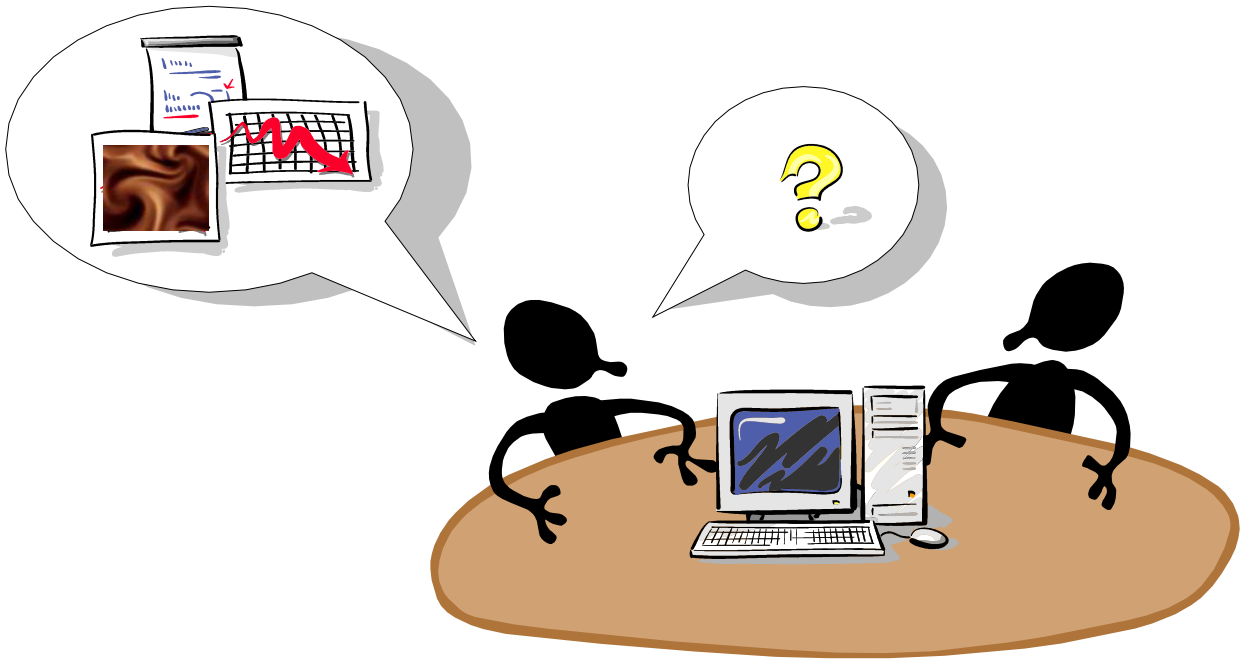


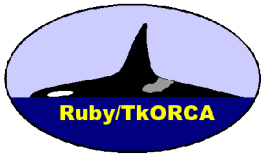
開発するフレームワークを使うと...

A: 『こんな GUI で研究データ分析用ツールを作ってみたんだけど，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』

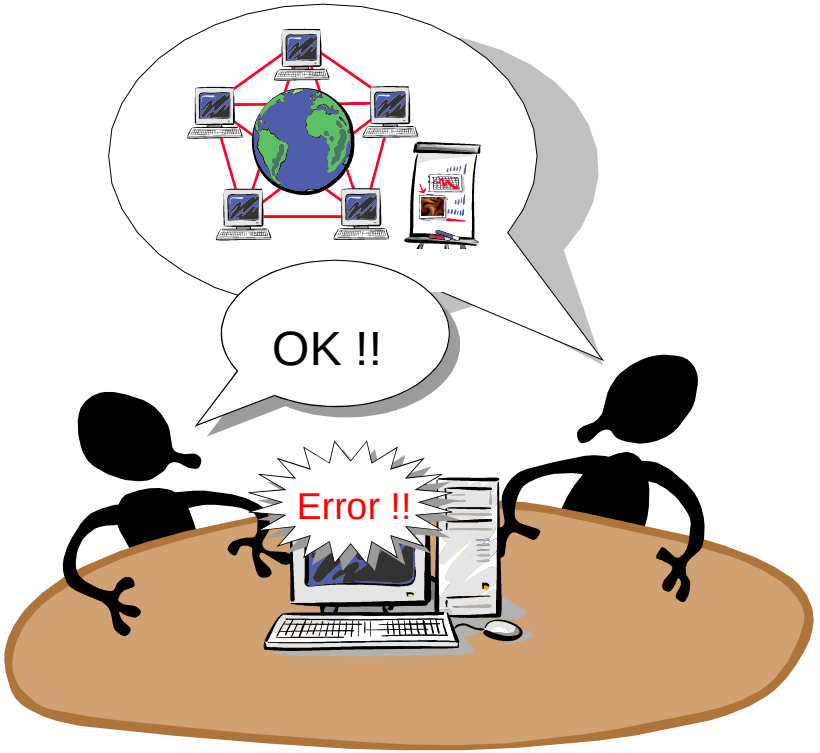


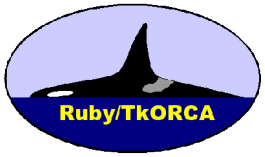


開発するフレームワークを使うと...

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，テスト環境で実行してっと...
あ，セキュリティエラーか．データディレクトリにアクセスしている部分みたい』



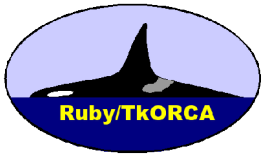


開発するフレームワークを使うと ...

ローカルのウィンドウシステムで実行テストが可能

- ローカルで動く GUI アプリをそのまま利用可能
- 実行テストのためにサーバを稼働させる必要なし
- 公開時と同じに sandbox 内で実行
- 公開時と同じ表示をローカルに表示
- 実行テストで最低限のセキュリティは確保可能

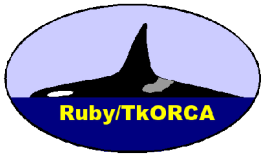
→ 開発速度や効率の向上



開発するフレームワークを使うと...

B:『その程度ならソースはそのままにして，指定したディレクトリだけはアクセスを許可するように監視スクリプトに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...』

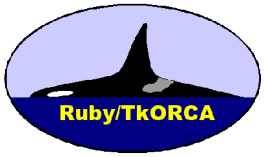


開発するフレームワークを使うと ...

致命的問題でなければ公開対象のソースは修正不要

- 公開対象は，監視スクリプトの監視下で実行
- 一般的な問題は監視スクリプト上の設定で対処可能
- 複雑なものでなければ，監視処理を記述し追加することでの対処も可能

→ 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



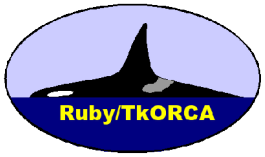
開発するフレームワークを使うと...

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！』

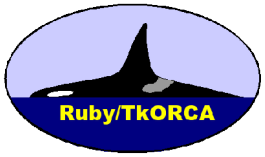


開発するフレームワークを使うと ...

容易な公開設定と柔軟なクライアント対応

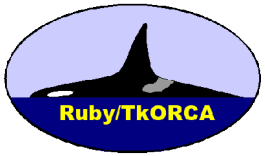
- テスト環境は公開時環境と同等であるため，公開や更新の作業はサーバでの登録を書き換える程度
- 必要なら，サーバでのローカル実行も可能
- thin client であるため，パソコンだけでなく PDA レベルの機器でもクライアントとなることが可能
- サーバ上のアプリを更新してもクライアントの更新は必要なし

→ ニーズに対しての極めて高い即応性

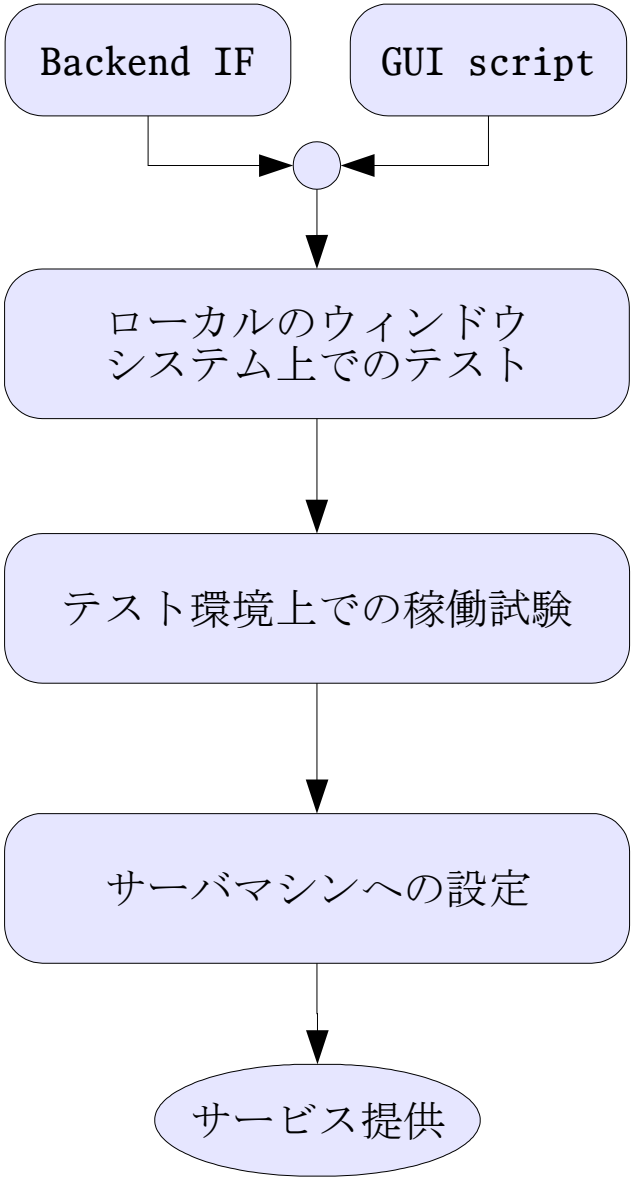


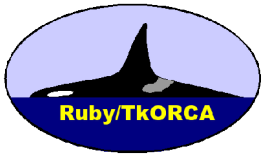
なぜそのようなものが欲しいのか？

- せっかく作ったGUIなんだから，そのままにローカルでも外からでも使えるようにしたい
- 外に公開するためにGUIの操作性を犠牲にしなきゃならないなんて，冗談でしょ
- 外から使いたくなるかどうかわからないような段階から公開のことを考えて作成するなんて，労力の無駄
- 公開用に新たに作り直すなんて，そんな面倒なことはしたくない
- ローカルのGUIならグラフ化なんかもライブラリで一発なのに，外向けには画像化などの細工がいるなんて馬鹿馬鹿しい
- Ajax？ Coolなのが作れるのはわかるけど，クライアントの能力への依存が大きいし，サーバとクライアントとのそれぞれ用にスクリプトを作ったりとお手軽じゃないよなぁ
- ローカルでちょっと使うだけの時にまで，重たい環境や重たいブラウザなんて立ち上げたくない
- 外に公開するときは，動作状況の監視くらいはしたいよね

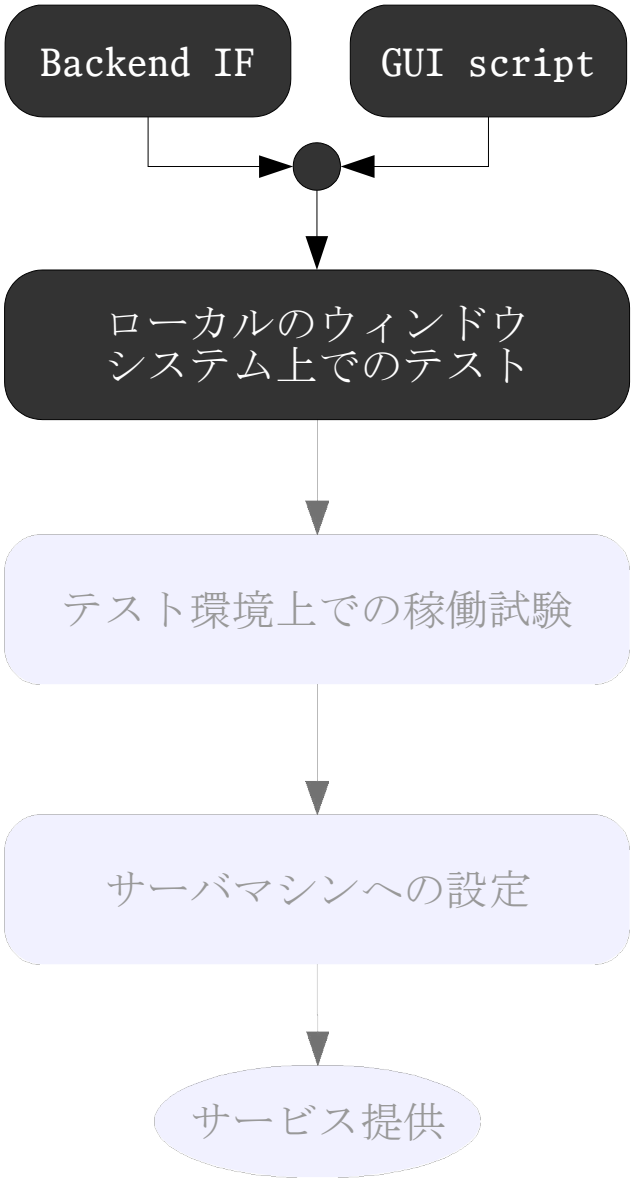

Ruby/TkORCA

開発工程と要求スキル


Backend IF
GUI script
ローカルのウィンドウ
システム上でのテスト
テスト環境上での稼働試験
サーバマシンへの設定
サービス提供

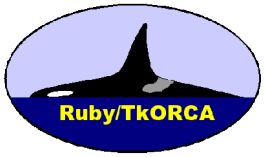


開発工程と要求スキル

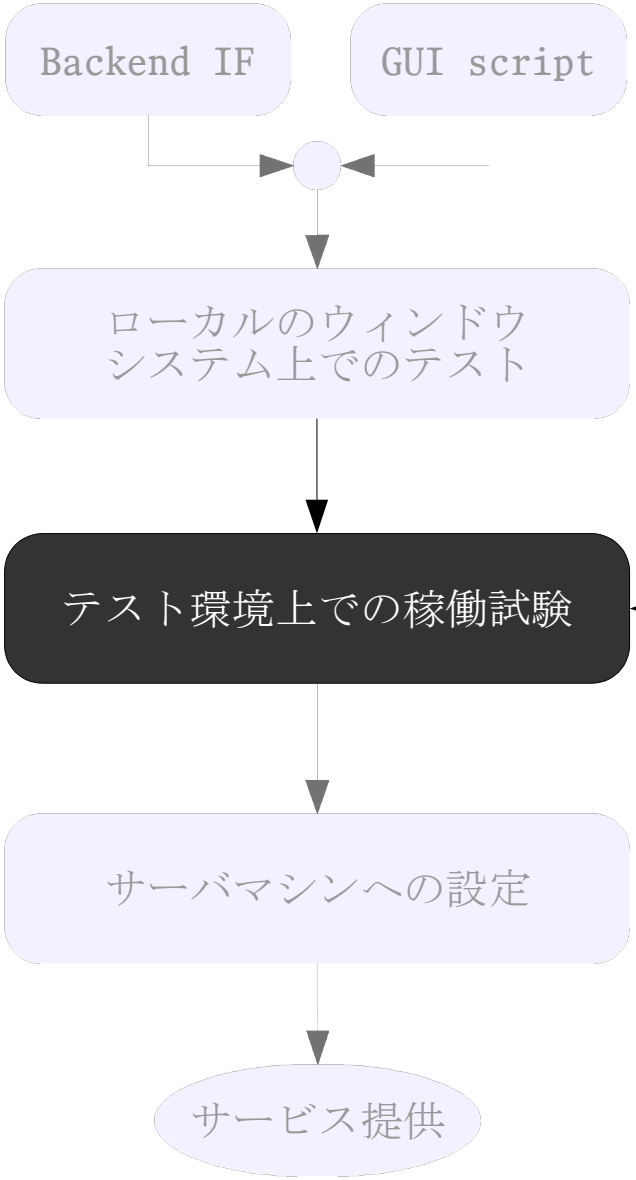


ローカルで稼働する GUI の作成

- GUI の最表層部は Ruby/Tk
  ↓
  Ruby および Ruby/Tk (Tcl/Tk でも可) の最低限の知識が必要
- ローカル用 GUI を普通に作るのと同じ
- 本フレームワークのための特殊な GUI 作成スキルは不要
- バックエンドとの IF 部で sandbox での稼働を考慮しておくと後で楽



開発工程と要求スキル

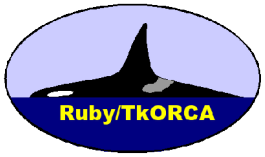


公開時と同等の環境での実行試験

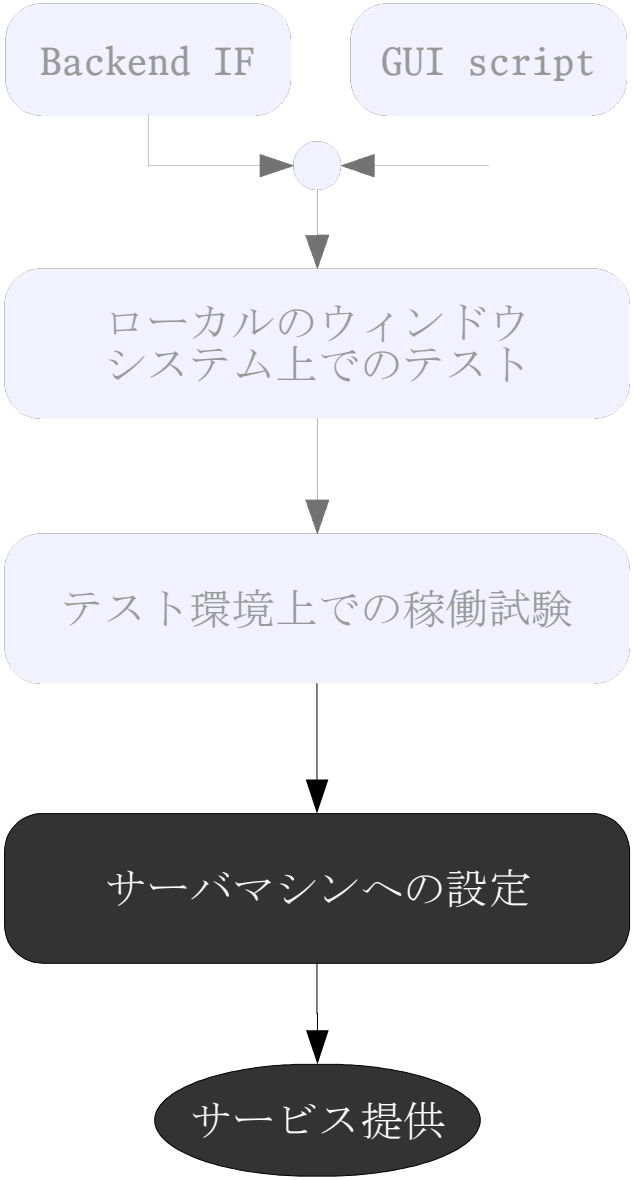
- 安全性を確かめるためのテスト
- Ruby のセーフレベルの制約や決まりごとを理解していることが望ましい
- 特殊な監視や制御が必要なら，監視スクリプトを追加することも可能

監視スクリプト

- フレームワークにおける監視や制御の設定項目や API の理解は必要
- 監視や制御によっては Ruby/Tk での Tk インタープリタ操作スキルも必要



開発工程と要求スキル

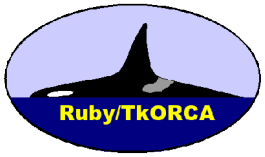


サービスの提供

- VNC などの基盤部分についての知識は必須ではない（隠蔽されるため）
- 一般的なネットワークサービスと同様のため，ネットワークに関する知識を持つことが望ましい
- サーバ設定ツールは提供の予定
- 例えば外部からの要求に対して

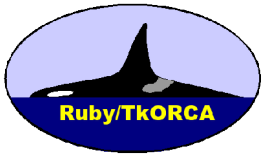
```
起動スクリプト [-a <監視スクリプト>]
    <GUI アプリ> [ <GUI アプリ> ... ]
```

などと起動するようにする程度の作業

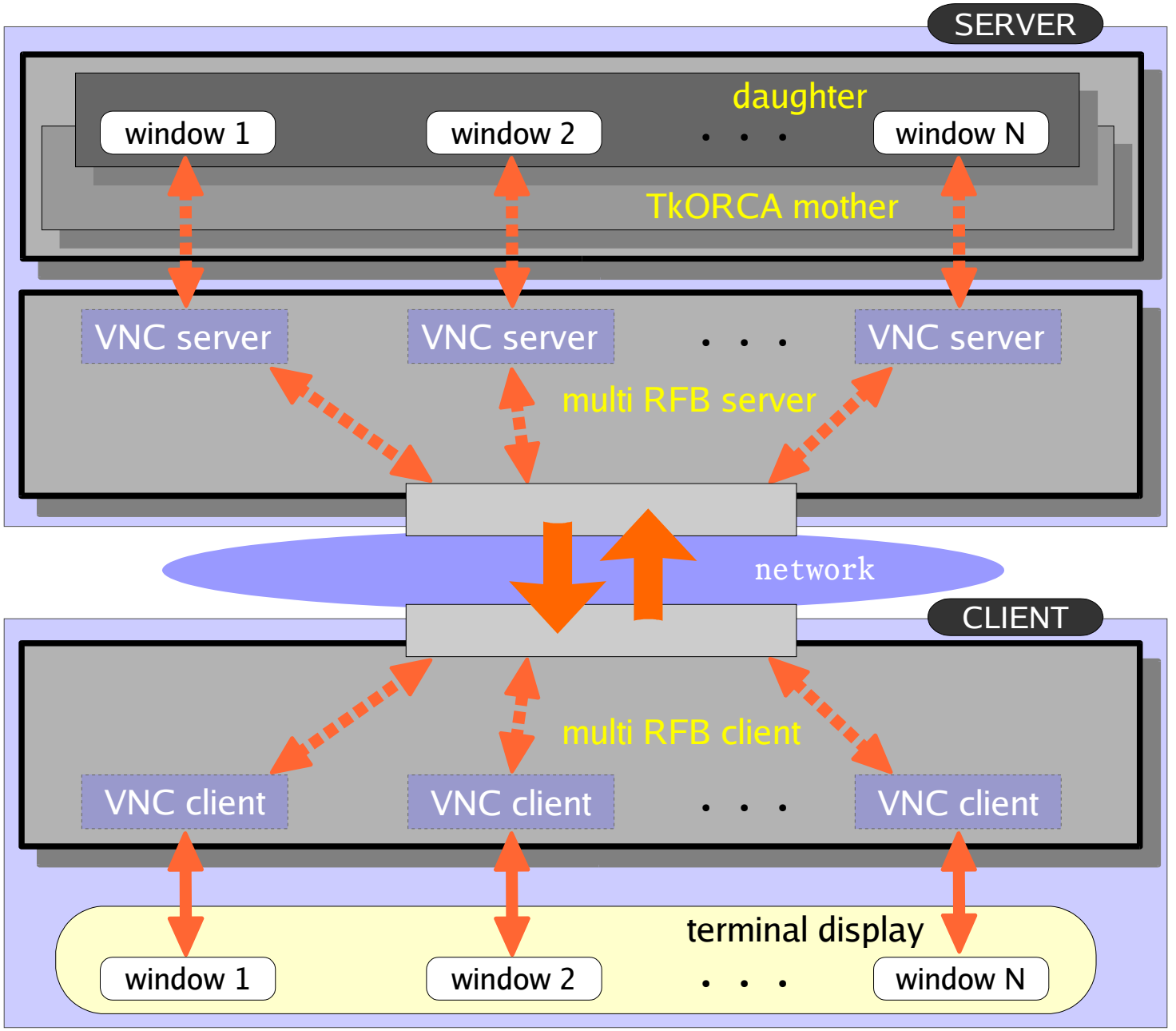


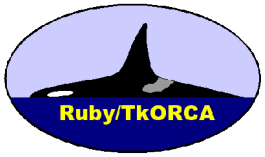
将来は？

- マルチウィンドウ RFB サービス
  - 1 mother/1 daughter での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する
- サーバ・クライアント間データ転送機能の提供
  - クライアント側アプリの強化は必要となるが，クライアントへの性能要求を強めるほどではないはず
- Tk の描画処理部分と RFB サービスとの一体化
  - GUI 部が pure Ruby/Tk なら，１プロセスだけでサービス提供可能
  - サーバ負荷の軽減に直結
- セッションの継続 (GUI の永続化)
  - 継続可能性は条件付きとなるが，現在の GUI の構造を調査して保存，後にそれに基づいて再構築することで永続化を図る



マルチウィンドウ RFB サービス





将来は？

- マルチウィンドウ RFB サービス
  - 1 mother/1 daughter での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する
- サーバ・クライアント間データ転送機能の提供
  - クライアント側アプリの強化は必要となるが，クライアントへの性能要求を強めるほどではないはず
- Tk の描画処理部分と RFB サービスとの一体化
  - GUI 部が pure Ruby/Tk なら，１プロセスだけでサービス提供可能
  - サーバ負荷の軽減に直結
- セッションの継続 (GUI の永続化)
  - 継続可能性は条件付きとなるが，現在の GUI の構造を調査して保存，後にそれに基づいて再構築することで永続化を図る