Schnelle modulare Synthese auf Mehrkern-Architekturen

GPN7

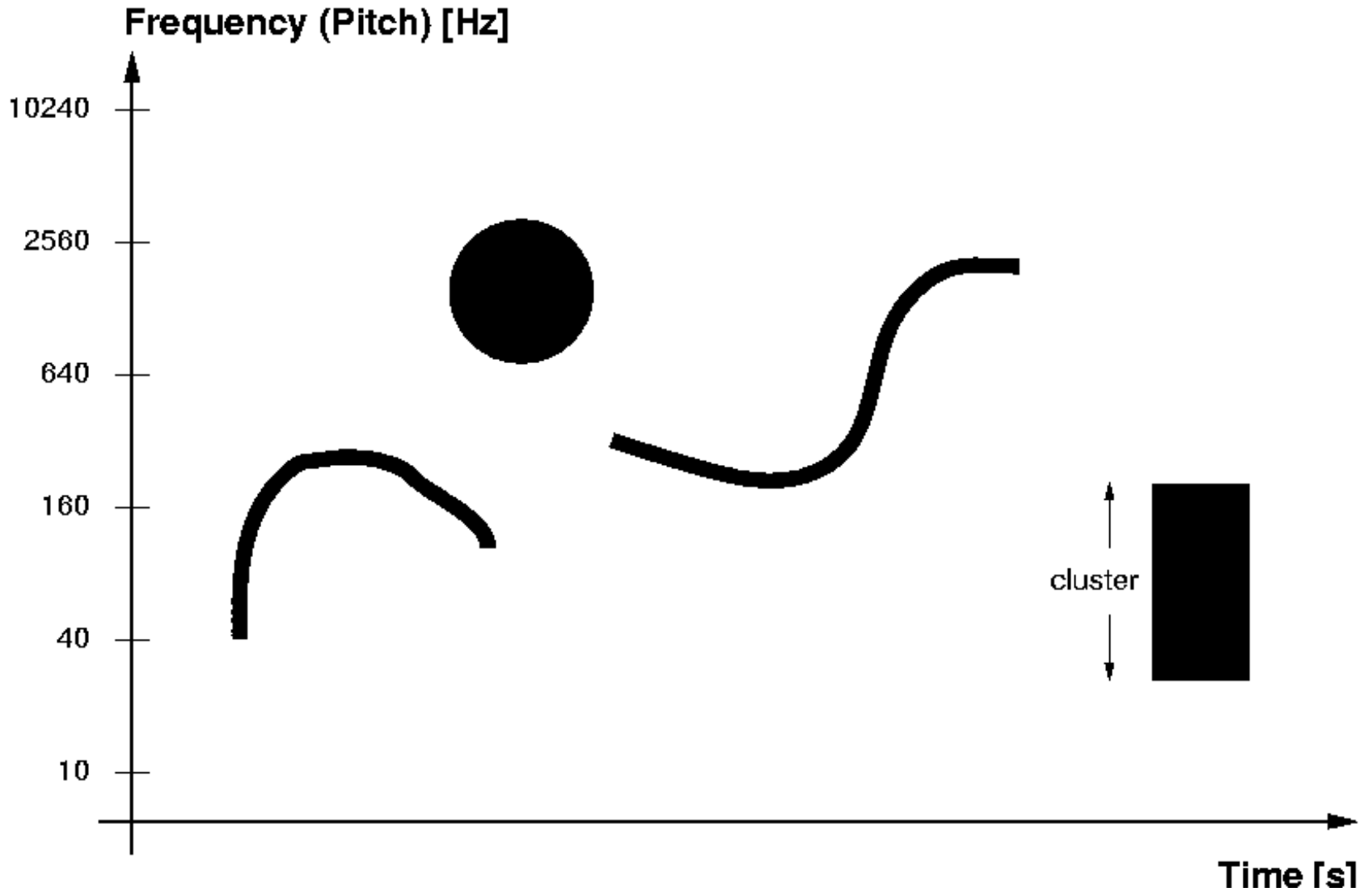
So, 06.07.2008

Jürgen Reuter

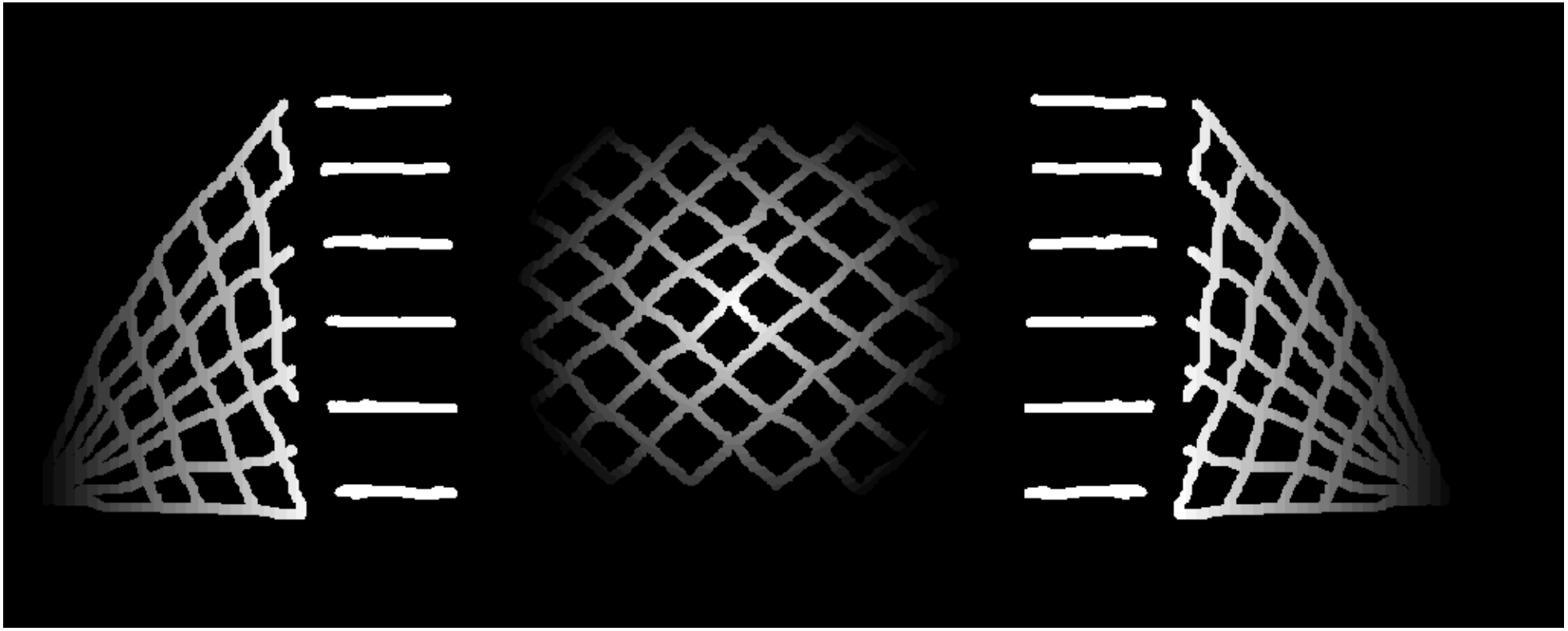
Ausgangspunkt: SoundPaint

- Integriere Entwurf von Klängen & Komposition
- Ziel: Ausdrucksvollere elektronische Musik
- Grafisches Komponieren
- Einfache, intuitive Bedienung

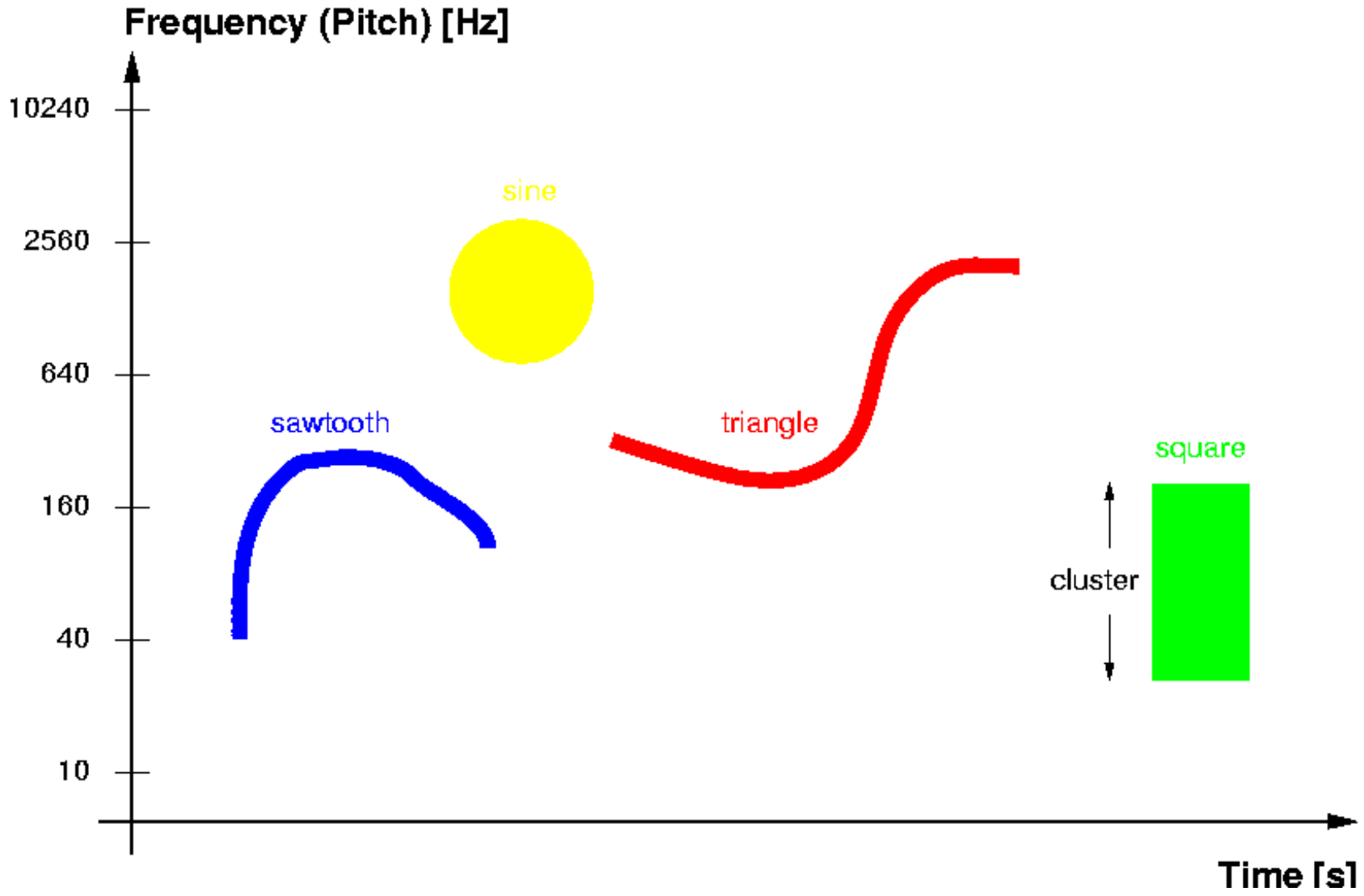
Additive Synthese



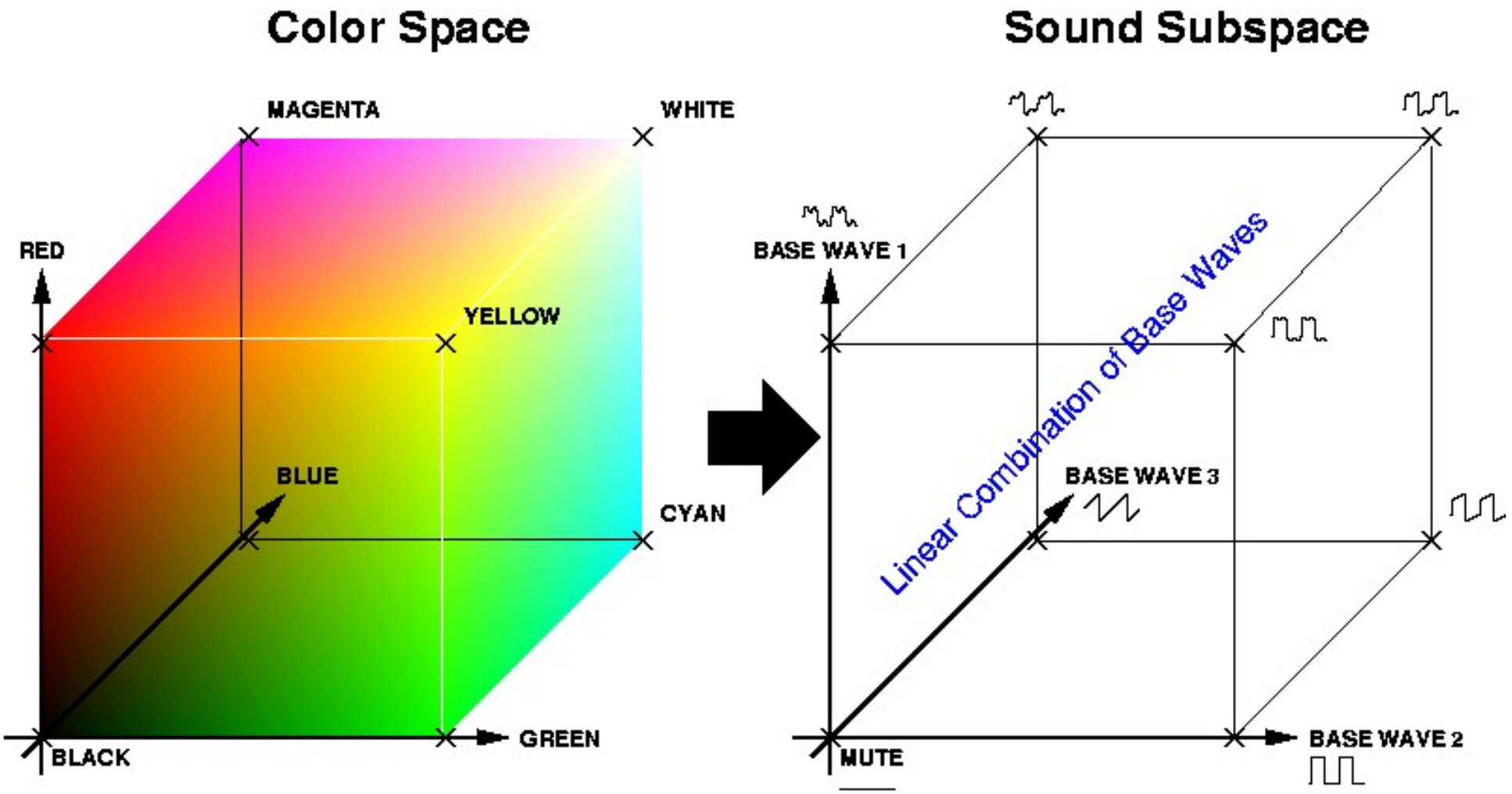
Klangbeispiel (1998)



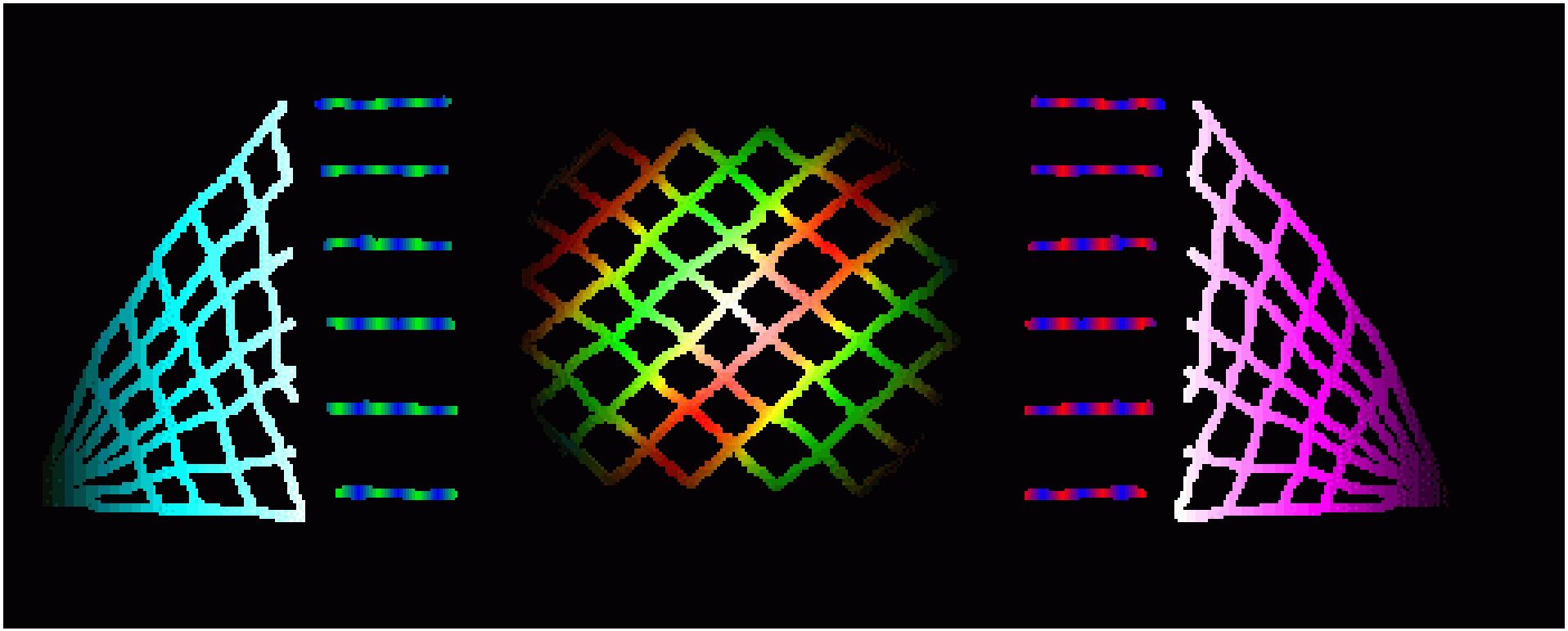
Farbe → Klang (2003)



Umsetzung Farbe → Klang



Klangbeispiel (2003)

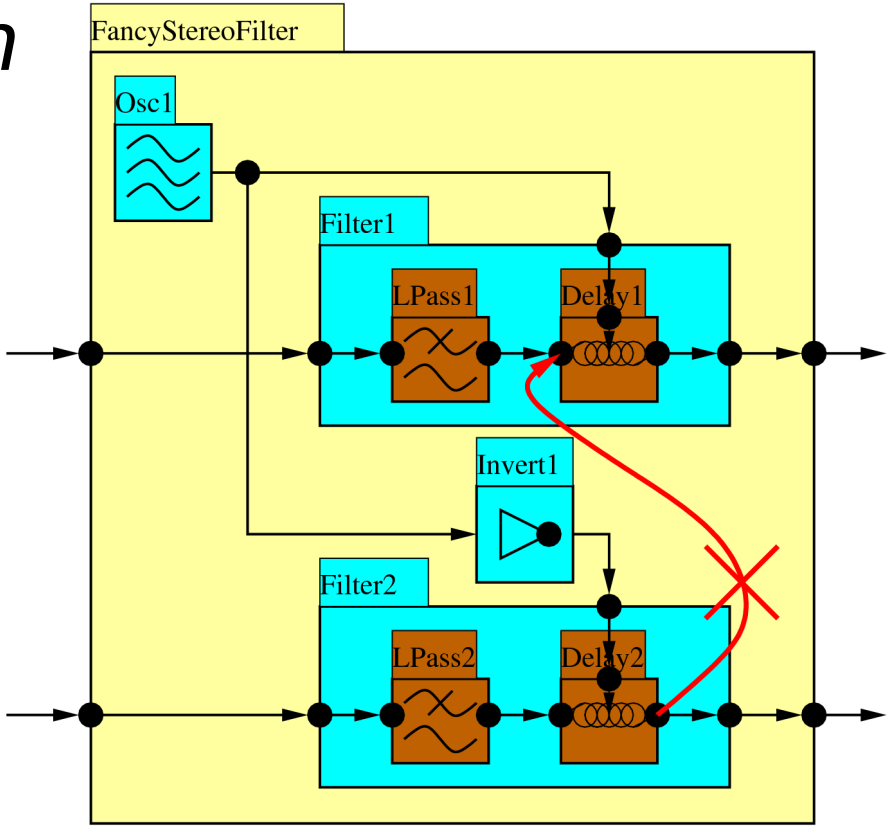


Mehrkern-Prozessoren

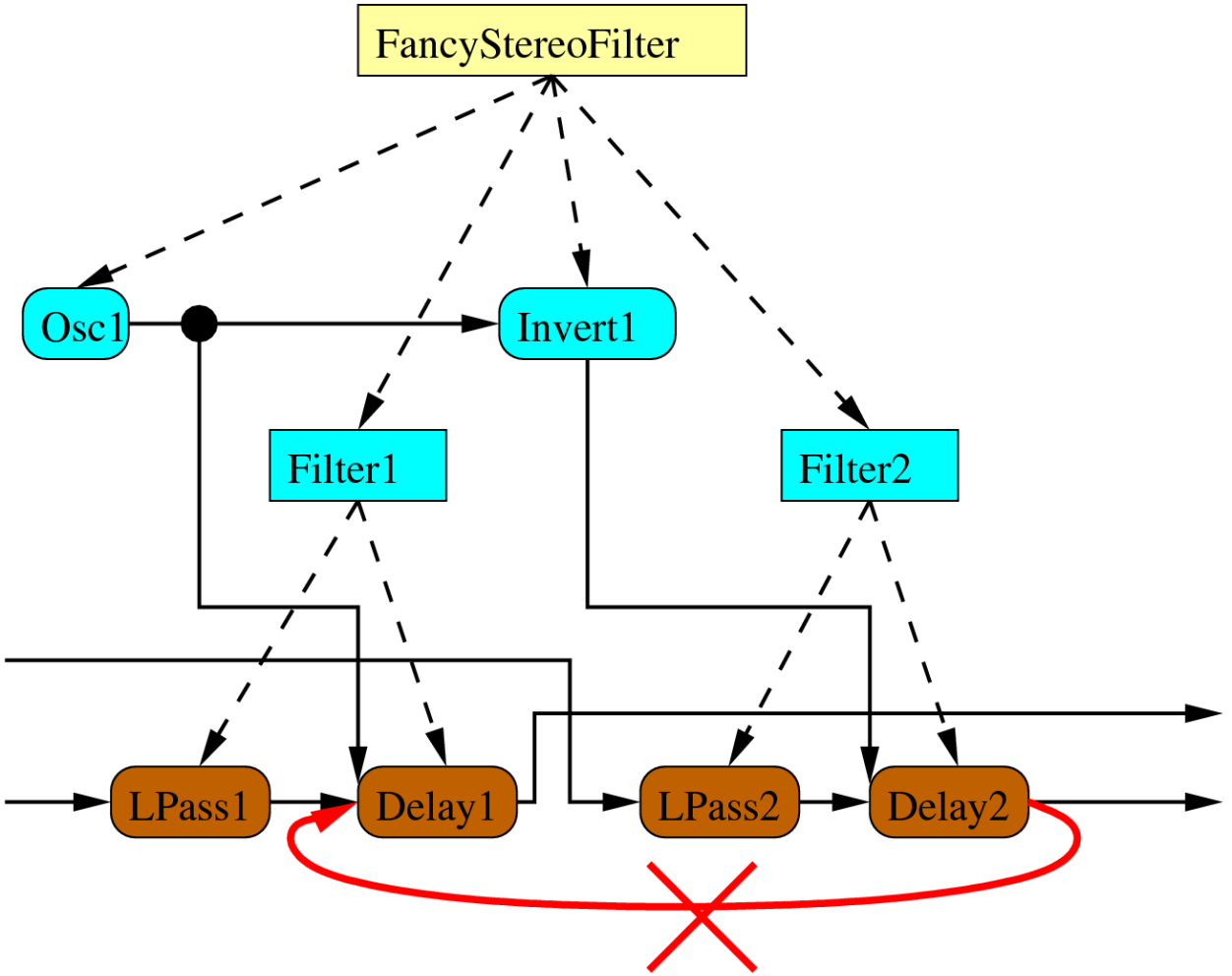
- Taktfrequenzen stagnieren
- Mehrkern-CPUs durchdringen den Markt
- Ideal für mehrfädige rechenintensive Aufgaben
- Anwendungen meist noch nicht parallelisiert
- Gute Unterstützung im Linuxkern aus Erfahrung mit SMP erwachsen
- Hier: *Wie kann modulare Synthese parallelisiert werden?*

Modulares Prinzip der Synthese

- Hierarchie von *Modulen*
- *Eingänge*
- *Ausgänge*
- *Primitive* Module
- *Zusammengesetzte* Module
- Keine Verbindung zwischen *verschiedenen* Untermodulen



Repräsentation als Modulbaum



Zeitmodell

- Ziel: Sample-synchroner Betrieb
  - Ein Zeitschritt pro Sample
- Berechne Modulausgänge aus Moduleingängen (und ggf. innerem Modulzustand)
- Übertrage Samples an Modulausgängen zu den verbunden Moduleingängen

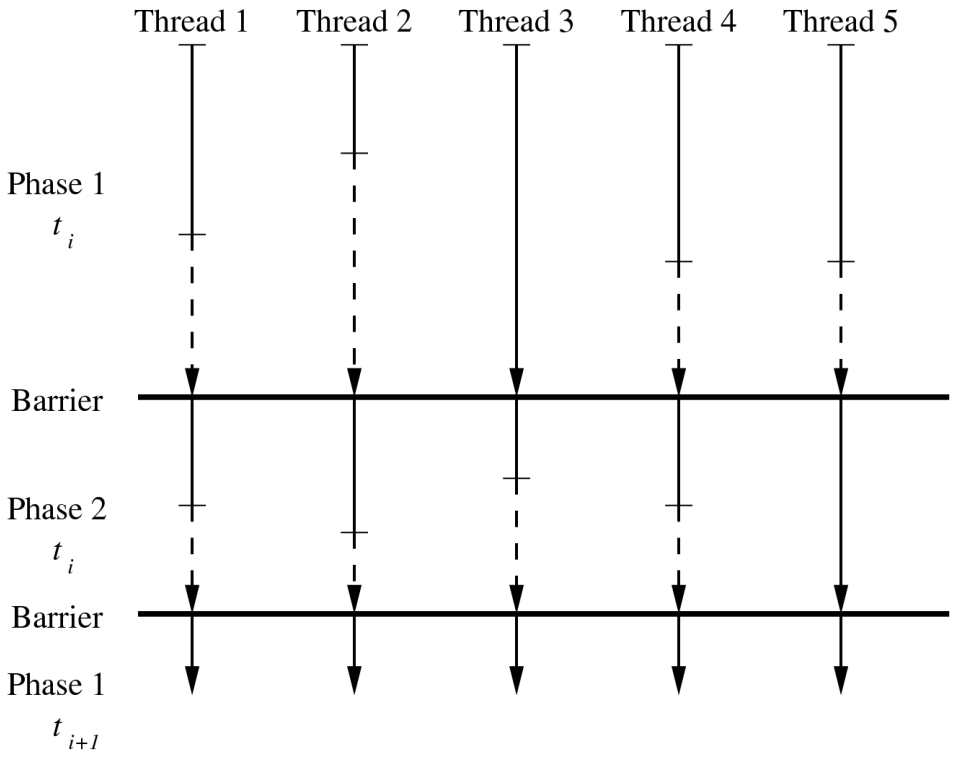
2 Phasen: Berechnen / Aktualisieren

- Mehrfädigkeit nutzen
- Sample-synchrone Aktualisierung
- Aber: Abhängigkeiten zwischen Modulen
- => Reihenfolge der Aktualisierung wichtig
- Trennen in Phasen *berechnen & aktualisieren*

```
while (true) do {
  // Phase 1: Compute
  for all modules do {
    compute outputs for next time step
    in terms of other module's outputs,
    but keep results private to this
    module
  }
  // Phase 2: Update
  for all modules do {
    publish outputs to other modules
  }
}
```

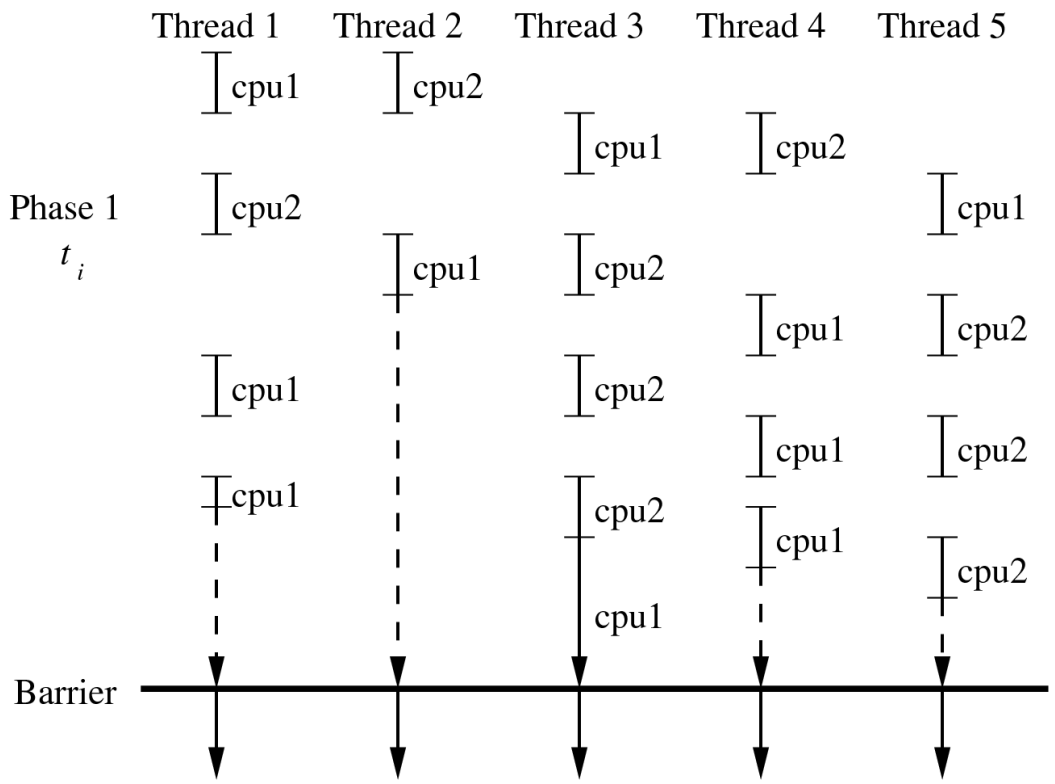
Barrieren-Synchronisation

- Starte Phase 2 erst, wenn Phase 1 für alle Fäden abgeschlossen
- Und umgekehrt
- => Verwende Barrieren, um Fäden zu synchronisieren



Round-Robin Scheduling

- Pro Modul einen Faden verwenden?
- Schlechte Idee:
  - OS verteilt Fäden auf CPUs
  - => Zahlreiche Taskwechsel
- Lösung: pro Faden mehrere Module bearbeiten



Zuordnung Modul-Faden (1)

- Wie viele Fäden verwenden?
  - Wenige Fäden: schlechte Ausnutzung der Kerne
  - Viele Fäden: erhöhter Verwaltungsaufwand durch Taskwechsel
  - => Abwägung finden

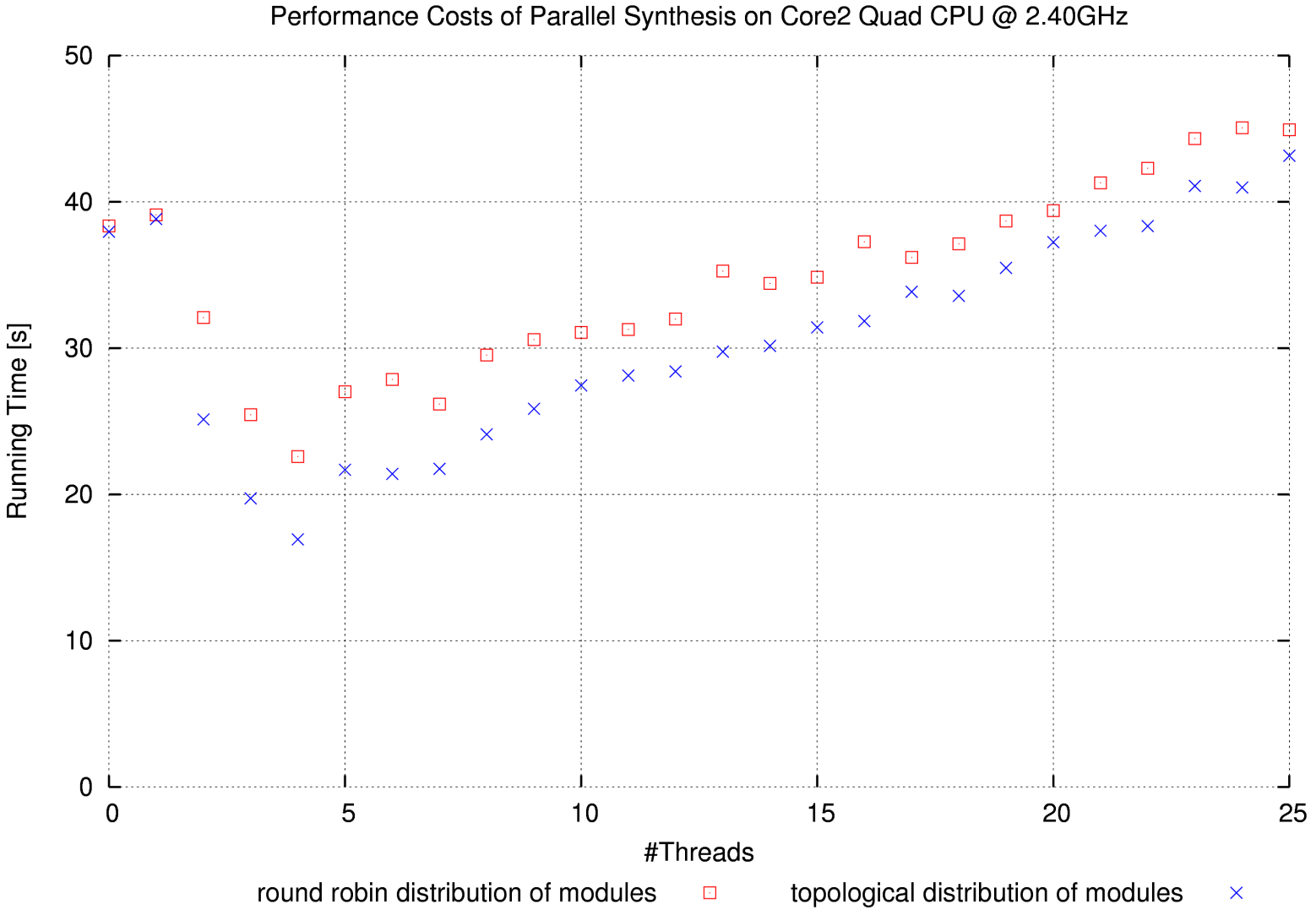
Zuordnung Modul-Faden (2)

- Welchem Faden welche Module zuordnen?
  - Hohe Datenlokalität => mehr Cache Hits
  - Gute Lastverteilung => Kurze Wartezeit an Barriere
- Hier zwei Ansätze
  - Round-robin (bzw. pseudo-zufällige) Zuweisung der Module zu Fäden => bessere Lastverteilung?
  - Zuweisung entsprechend der Repräsentation als Modulbaum => bessere Datenlokalität?

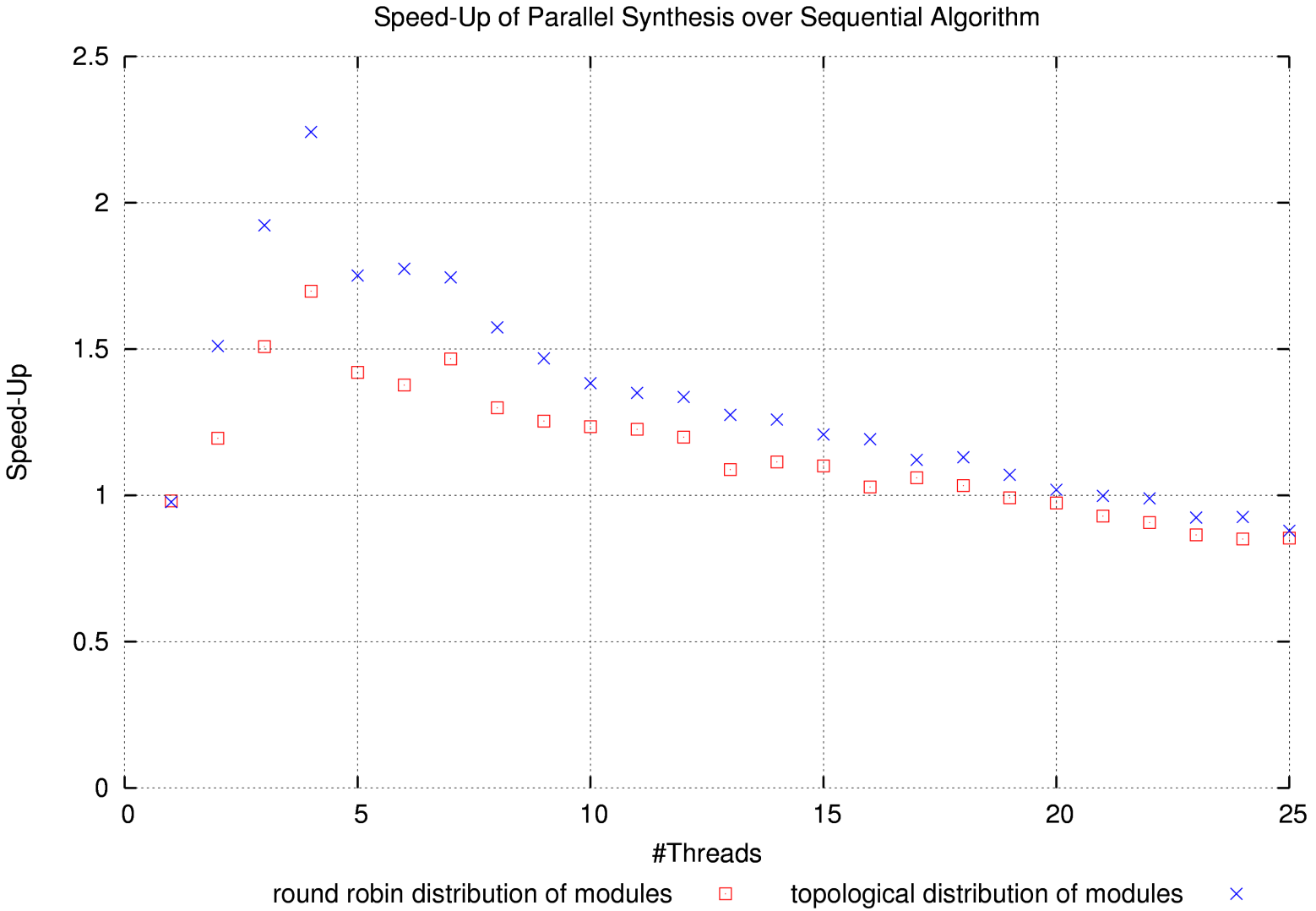
Auswertung

- Implementierung in Java
  - Anzahl der Fäden einstellbar
  - Java-Fäden 1:1 nativen Linux-Fäden zugeordnet
- Vergleich Zuordnung Module=>Fäden gemäß
  - Round-robin (pseudo-zufällig)
  - Topologisch geordnet gemäß Modulbaum
- Pseudo-Synth (Feld mit ~2000 Oszillatoren)
- Realer Synth (SoundPaint) mit ~1650 Modulen
- Hardware: Core Quad CPU Q6600 @ 2.40 GHz

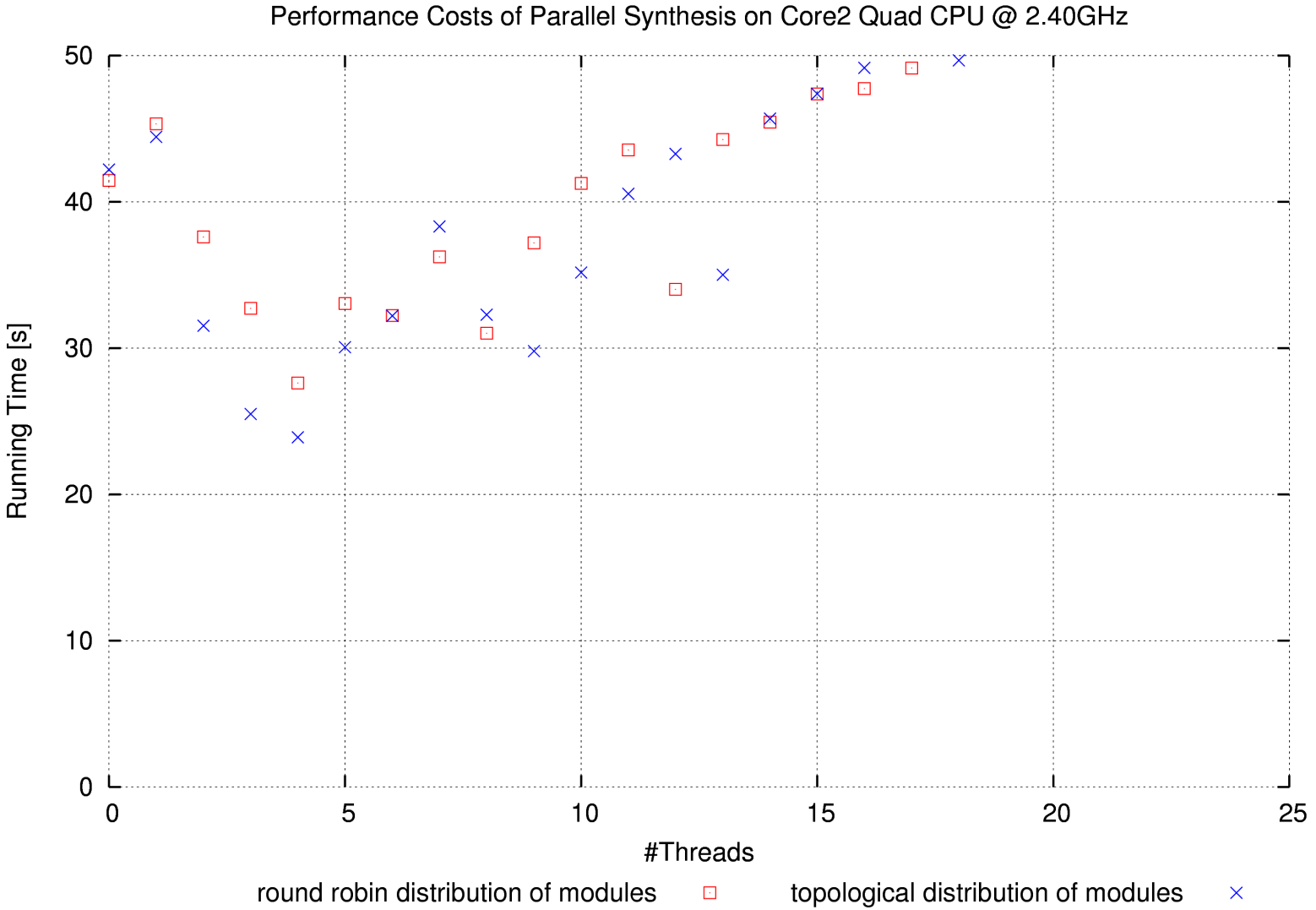
Pseudo-Synth Performance



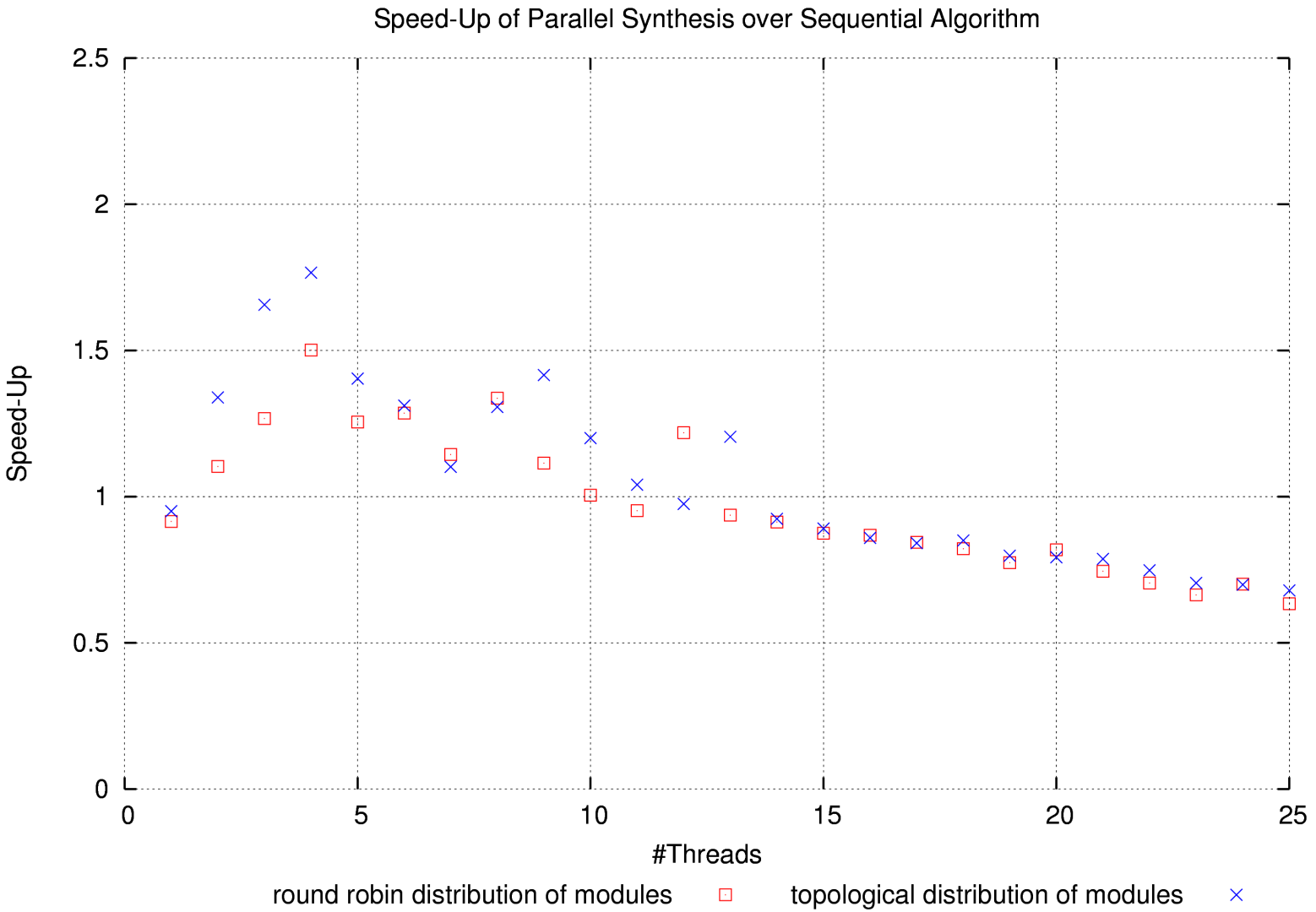
Pseudo-Synth Speed-Up



SoundPaint Performance



SoundPaint Speed-Up



Ergebnisse

- Nur geringfügiger Overhead des mehrfädigen Algo mit 1 Faden gegenüber sequenz. Algo
- Optimum @ 4 Fäden (=Anzahl CPU Cores)
- “Real-life” SoundPaint-Messung nicht ganz so deutlich im Ergebnis
  - Vielleicht wegen unregelmäßiger Berechnungszeiten?
- Modulzuordnung gemäß Modulbaum im Schnitt deutlich besser als pseudo-zufällige Verteilung

Zukünftige Arbeiten

- Ursache für höhere Performance der Verteilung gemäß Modulbaum unklar
  - Schlechte Datenlokalität der pseudo-zufälligen Verteilung der Module auf Fäden?
  - CPUs running idle at barriers?
- Gesamt-Speed-up noch verbesserungswürdig
  - Lastverteilung (Leerlaufende CPUs an Barrieren?)
  - Leerlaufende CPU: ggf. Samples vorausberechnen (z.B. bei Modulen ohne Eingangsabhängigkeiten)
  - Verschmelze lokale leichtgewichtige Module

Zusammenfassung

- Mehrere Fäden verwenden, um Mehrkern-CPUs zu nutzen
- Aber nicht zu viele Fäden (Overhead durch Taskwechsel!)
- => Unterstütze anpassbare Anzahl an Fäden
- Umsichtig Verteilung der Aufgaben an die Fäden
  - Datenlokalität (vermeide Cache Misses)
  - Lastverteilung (vermeide leerlaufende CPUs an Barrieren)

Call For Participation: Projekt ***microDJ***

- Bestehendes Musikstück nehmen
- In Schnipsel der Länge <50ms (entspricht Hörgrenze 20Hz) schneiden
- Schnipsel tragen Klang – nicht Ereignis
- Nach geigneter Metrik umsortieren
  - Klanglicher Abstand, z.B. FFT oder Wavelet-Trafo
  - Nachbarschnipsel im Original nach Umsortierung nicht mehr Nachbarn
  - Musikalische Zielparameter (z.B. vorgegebener Lautstärkeverlauf)
- Wieder zusammenfügen (ggf. mit Überblenden)

Fragen?

- Code (noch sehr instabil, daher eher zum Reinschauen als zum produktiven Einsatz): **www.soundpaint.org/modsynth**
- Relevanter Code für Barrierensynchronisation und Modulverteilung z.Z. in der Java-Klasse **org.soundpaint.modsynth.syntest.Master**